Contents

[1. Các lợi ích của RegEx 3](#_Toc78395865)

[2. Bắt đầu sử dụng Regular Expression 9](#_Toc78395866)

[3. Tóm lại 10](#_Toc78395867)

[4. Ví dụ đơn giản 11](#_Toc78395868)

[4.1. Tìm chuỗi số 11](#_Toc78395869)

[4.2. Tìm ngày tháng chính xác 12](#_Toc78395870)

[4.3. Cách bắt lấy chuỗi bất kỳ 13](#_Toc78395871)

[4.4. Lấy một chuỗi nhưng không chứa 1 ký tự đặc biệt nào đó 14](#_Toc78395872)

[4.5. Lấy một chuỗi nhưng không chứa 1 chuỗi con đặc biệt nào đó 14](#_Toc78395873)

[4.6. Bắt 1 chuỗi bất kỳ nằm ở nhiều dòng (mutiline) 16](#_Toc78395874)

[4.7. Check một chuỗi ngày tháng bất kỳ có hợp lệ hay không tính theo năm nhuận 17](#_Toc78395875)

[4.8. Check 1 chuỗi ngày tháng phải có cả ngày và giờ 18](#_Toc78395876)

[4.9. Check 1 chuỗi bất kỳ xem có phải ngày hoặc giờ hay không 19](#_Toc78395877)

[4.10. BTU1. Tìm ra các chuỗi là số điện thoại trong chuỗi sau. Chú ý một số loại ra một số chuỗi không phải là số điện thoại 20](#_Toc78395878)

[4.11. BTU2. Tìm ra chuỗi là địa chỉ email trong chuỗi sau, chú ý loại bỏ các chuỗi k phải địa chỉ emai thực sự. 22](#_Toc78395879)

[4.12. BTU3. Tìm cách lấy ra và ngăn cách chuỗi số sau thành có phần trăm phần nghìn. 24](#_Toc78395880)

[4.13. BTU4. Lấy ra các chuỗi ngăn cách bởi dấu phẩy theo định dạng của một file CSV như sau (chú ý là chuỗi bị xuống dòng): 25](#_Toc78395881)

[4.14. BTU5. Tìm và xóa đi toàn bộ các thẻ HTML trong đoạn text dưới đây: 26](#_Toc78395882)

[4.15. BTU6. Tìm cách loại bỏ các chuỗi console.log() do lập trình viên viết để test ra khỏi code trước khi bàn giao 27](#_Toc78395883)

[4.16. BTU7. Loại bỏ toàn bộ các đoạn comment code trong đoạn code sau đây: 28](#_Toc78395884)

[4.17. BTU8. Lấy ra các chuỗi là các URL đầy đủ trong đoạn text dưới đây: 29](#_Toc78395885)

[4.18. BTU9. Lấy ra toàn bộ các URL của một video Youtube từ đoạn text test sau đây 30](#_Toc78395886)

[4.19. BTU10. Lấy ra các đoạn text là giá trị màu RBG 31](#_Toc78395887)

[4.20. Độ mạnh của mật khẩu 32](#_Toc78395888)

[4.21. Mã màu Hex 32](#_Toc78395889)

[4.22. Xác thực địa chỉ Email 32](#_Toc78395890)

[4.23. Địa chỉ IPv4 32](#_Toc78395891)

[4.24. Địa chỉ IPv6 32](#_Toc78395892)

[4.25. Dấu phân cách hàng nghìn 32](#_Toc78395893)

[4.26. Thêm HTTP vào trước liên kết 33](#_Toc78395894)

[4.27. Lấy tên miền từ URL 33](#_Toc78395895)

[4.28. Sắp xếp các từ khóa bằng cách đếm số từ 33](#_Toc78395896)

[4.29. Tìm một chuỗi Base64 hợp lệ trong PHP 33](#_Toc78395897)

[4.30. Xác thực số điện thoại 34](#_Toc78395898)

[4.31. Khoảng trắng ở đầu và cuối 34](#_Toc78395899)

[4.32. Lấy nguồn ảnh 34](#_Toc78395900)

[4.33. Xác thực ngày trong định dạng DD/MM/YYYY 34](#_Toc78395901)

[4.34. Khớp ID Video Youtube 34](#_Toc78395902)

[4.35. Xác thực ISBN 35](#_Toc78395903)

[4.36. Kiểm tra mã bưu điện 35](#_Toc78395904)

[4.37. Tên người dùng Twitter hợp lệ 35](#_Toc78395905)

[4.38. Số thẻ tín dụng 35](#_Toc78395906)

[4.39. Tìm thuộc tính CSS 35](#_Toc78395907)

[4.40. Loại bỏ chú thích HTML 36](#_Toc78395908)

[4.41. URL trang cá nhân Facebook 36](#_Toc78395909)

[4.42. Kiểm tra phiên bản Internet Explorer 36](#_Toc78395910)

[4.43. Bóc tách giá 36](#_Toc78395911)

[4.44. Phân tích tiêu đề Email 36](#_Toc78395912)

[4.45. Khớp một loại tập tin cụ thể 37](#_Toc78395913)

[4.46. Khớp một chuỗi URL 37](#_Toc78395914)

[4.47. Thêm rel="nofollow" vào các liên kết 37](#_Toc78395915)

[4.48. Khớp Media Query 37](#_Toc78395916)

[4.49. Cú pháp tìm kiếm Google 37](#_Toc78395917)

[4.50. Khớp một Tên người dùng 38](#_Toc78395918)

[4.51. Khớp một Mật khẩu 38](#_Toc78395919)

[4.52. Khớp một Giá trị Hex 38](#_Toc78395920)

[4.53. Khớp một Slug 39](#_Toc78395921)

[4.54. Khớp một Email 39](#_Toc78395922)

[4.55. Khớp một URL 40](#_Toc78395923)

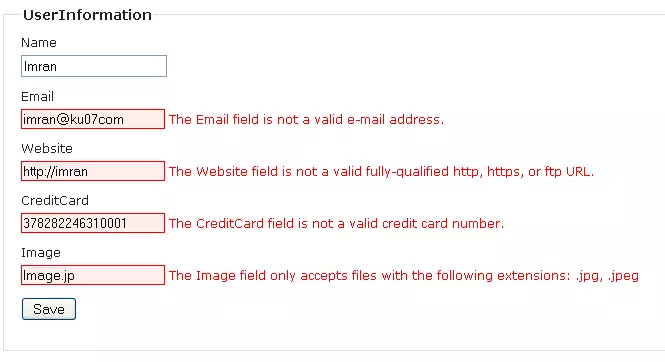
[4.56. Khớp một Địa chỉ IP 41](#_Toc78395924)

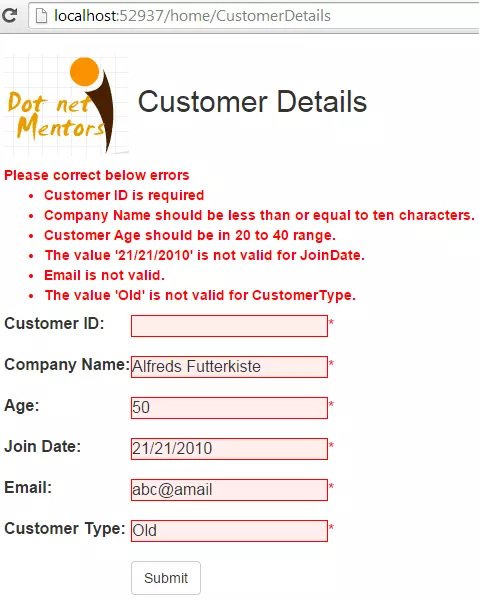
[4.57. Khớp một Thẻ HTML 41](#_Toc78395925)

Regular Expression

# Các lợi ích của RegEx

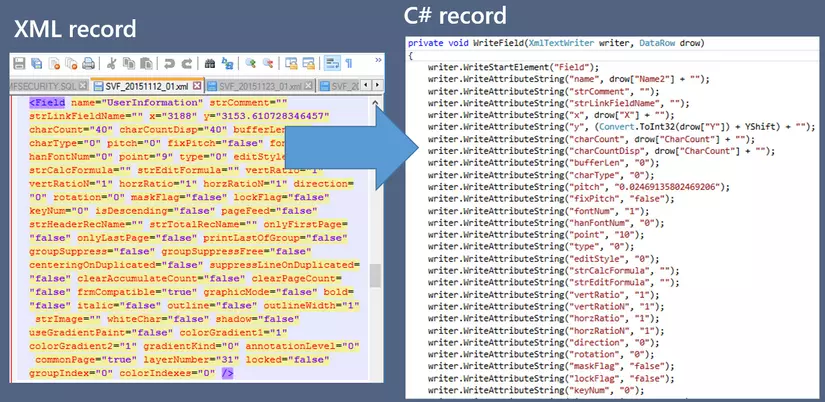
* Bạn cần xử lý validate (kiểm tra tính hợp lệ) các trường dữ liệu nhập vào ô Text





* Cần copy và paste rất nhiều text từ chỗ này sang chỗ kia (không copy thuần mà copy và chỉnh sửa)

Ví dụ từ XML sang C#



hoặc CSV, Excel sang C#

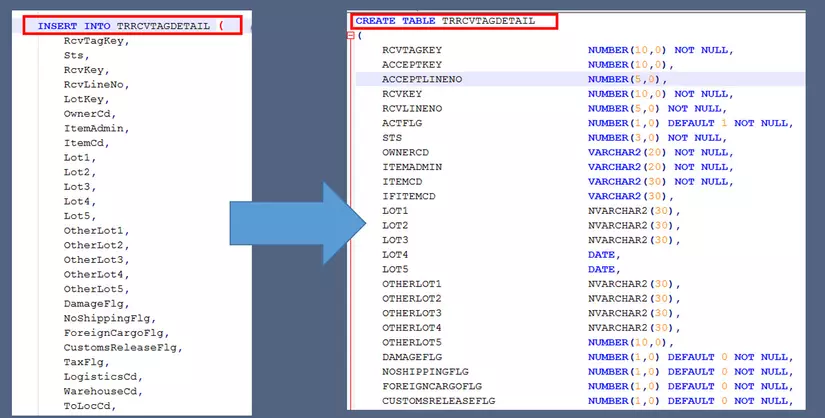


* Tìm kiếm một chuỗi nằm ở nhiều dòng trong Visual Studio hoặc Notepad++

Ví dụ tôi cần tìm và xóa bỏ (replace all) các comment như thế này ở code cũ.

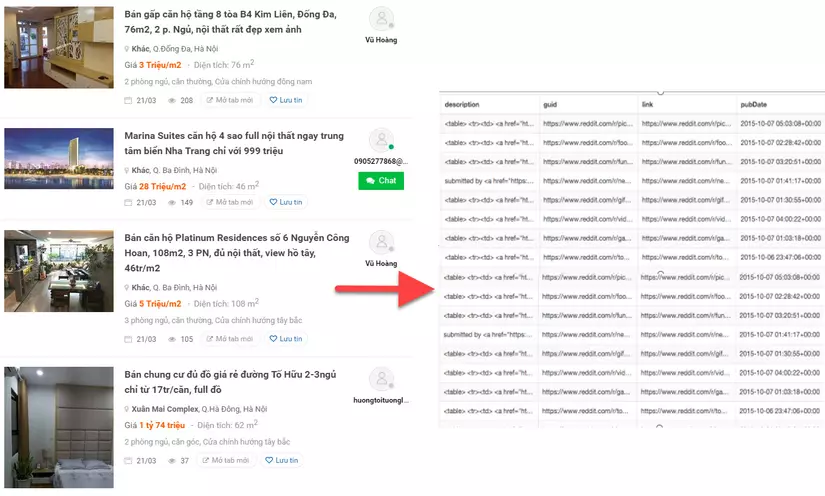


Kiểm tra bằng câu lệnh SQL xem đã insert đủ các trường vào DB hay chưa?



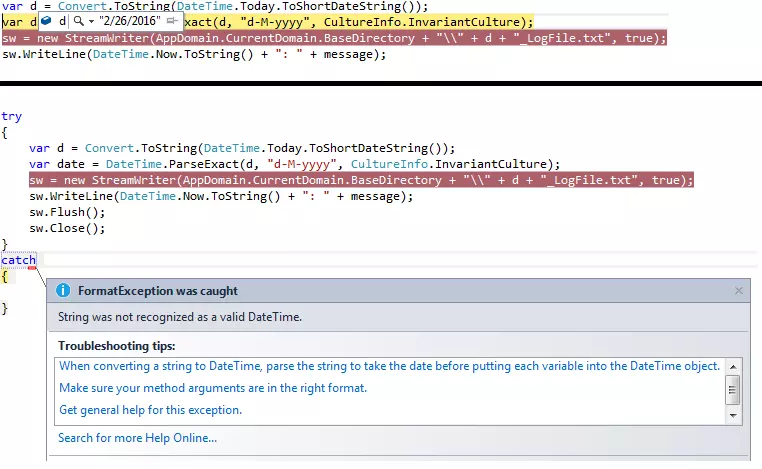
* Cần bóc tách dữ liệu của một trang web.

Ví dụ extract (crawling) tách lấy dữ liệu từ một trang web để lưu lại vào cơ sở dữ liệu của bạn



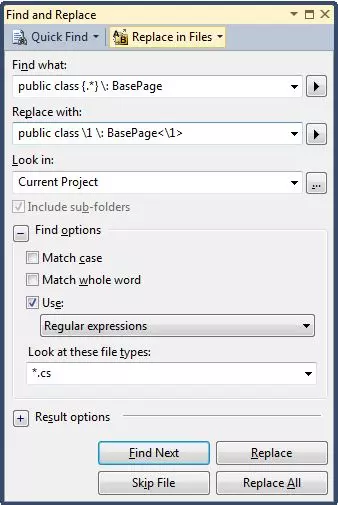
* Cơn ác mộng đọc một chuỗi string từ DB và cố gắng chuyển đổi nó sang DateTime

Chuỗi ngày tháng lưu vào CSDL rất đa dạng. Rất khó để dùng hàm DateTime.TryParse() để tự động chuyển đổi một chuỗi thành Datetime ngon lành cành đào.



* Tìm kiếm một chuỗi, lưu chuỗi đó lại và lấy chuỗi đã lưu chèn vào chỗ nào đó

Hay nói cách khác, ta cần thay một chuỗi bằng một chuỗi khác, trong chuỗi mới đó lại có chứa cả chuỗi vừa bị thay thế. Như ví dụ dưới đây sẽ cho các bạn thấy. Ta cần tìm ra ClassName, nhưng không phải thay thế ClassName đó bằng Class khác mà ta sẽ nhét nó vào 2 vị trí thay vì chỉ có 1 vị trí như hiện tại.



Regular Expression sinh ra là để giúp cho cuộc đời của bạn bớt khổ. Hãy tưởng tượng xem bạn phải nai lưng ra copy-paste bao nhiêu code. Bạn hoàn toàn có thể download các tool về để replace giúp bạn. Nhưng bạn cần bao nhiêu tool cho đủ đây.

Thay vào đó, bạn hãy thử học RegEx. Bộ cú pháp này sẽ giúp cho bạn thao tác với chuỗi như dao chém chuối. Như ta thấy các ví dụ bên trên, tất cả đều là XỬ LÝ CHUỖI ví dụ Cut Copy Paste hoặc Replace chuỗi. RegEx là ngôn ngữ giúp xử lý chuỗi rất mạnh.

RegEx không phải là một ngôn ngữ lập trình. Nó chỉ là một BỘ CÚ PHÁP dùng để bắt chuỗi. Nhưng nó cực kỳ phổ biến và bất kỳ ngôn ngữ lập trình nào cũng hỗ trợ. Nó có cả trăm ngàn ứng dụng và công cụ ăn theo.

# Bắt đầu sử dụng Regular Expression

Để bắt đầu sử dụng và học cách dùng RegEx, mình hay dùng trang web https://regex101.com/.

Trước hết các bạn hãy đọc qua một lượt các quy tắc bắt chuỗi (matching) đơn giản nhất của RegEx

* [xyz] Tìm và so sánh tất cả ký tự nằm trong dấu ngoặc vuông và trùng khớp với 1 ký tự trong dấu ngoặc vuông. Ví dụ: [31] sẽ trùng khớp với 3 hoặc 1, [0123456789] sẽ trùng khớp với bất kỳ một ký tự nào trong khoảng từ 0 đến 9.
* [a-z] So sánh và trùng khớp với một ký tự nằm trong khoảng chỉ định. Ví dụ: [a-z] sẽ trùng khớp với một ký tự trong khoảng từ a đến z nằm trong chuỗi cần test. [0-9] sẽ trùng khớp với bất kỳ một ký tự nào trong khoảng từ 0 đến 9.
* [^xyz] So sánh và không trùng khớp với những ký tự nằm trong khoảng chỉ định. Dấu ^ (dấu mũ) nằm trong dấu ngoặc vuông là một dấu phủ định. Ví dụ: [^a-z] sẽ không trùng khớp với tất cả các ký tự nằm trong khoảng từ a đến z.
* ^ Trùng khớp với phần đầu của chuỗi đích. Ví dụ: ^a sẽ trùng khớp với chữ a trong chuỗi abc, ^\w+ sẽ trùng khớp với chữ đầu tiên – chữ "the" của chuỗi "The quick brown fox jumps over the lazy dog".
* $ Trùng khớp với phần cuối của chuỗi đích. Ví dụ: c$ sẽ trùng khớp với chữ c trong chuỗi abc, \w+$ sẽ trùng khớp với chữ cuối – chữ "dog" của chuỗi "The quick brown fox jumps over the lazy dog".
* + Trùng khớp với 1 hoặc nhiều lần ký tự đứng trước nó. Ví dụ \d+ sẽ chỉ trùng với chuỗi có từ 1 con số trở lên.
* \* Trùng khớp với 0 hoặc nhiều lần ký tự đứng trước nó. Ví dụ \d\* sẽ trùng với chuỗi có chứa 1 chữ số hoặc k có chữ số nào cũng đc.
* ? Trùng khớp với 0 hoặc 1 lần ký tự đứng trước nó. Tương tự như \* nhưng nó lại chỉ nhân lên 1 lần. \* thì nhân lên nhiều lần.
* . Trùng khớp với 1 ký tự đơn bất kỳ ngoại trừ ký tự ngắt dòng (line-break) và cũng không lấy được ký tự có dấu (unicode). Ví dụ: . sẽ trùng khớp với ký tự a hoặc b hoặc c trong chuỗi abc. Nhưng. sẽ không bắt được các chữ ă hoặc ê.
* x{n} Trùng khớp đúng với n lần ký tự đứng trước nó. n là một số không âm. Ví dụ \d{2} sẽ bắt đc các số có 2 chữ số đứng liền nhau.
* x{n,} Trùng khớp với ít nhất n lần ký tự đứng trước nó. n là một số không âm.Ví dụ \d{2,} sẽ bắt đc các số có từ 2 chữ số trở lên đứng liền nhau.
* x{n,m} Trùng khớp với ít nhất n lần và nhiều nhất là m lần ký tự đứng trước nó. n và m là một số không âm và n <= m. Ví dụ: a{1,3} sẽ khớp với hah, haah, haaah nhưng không khớp với haaaah.
* x|y Trùng khớp với x hoặc y. Ví dụ: slow|fast sẽ khớp với chữ slow hoặc fast trong chuỗi đích.
* \b Trùng khớp với toàn bộ ký tự đứng trước nó. Ví dụ: hello\b sẽ trùng khớp với toàn bộ từ hello trong chuỗi hello world nhưng sẽ không khớp với chuỗi helloworld.
* \B Ngược lại với \b, \B sẽ không khớp với toàn bộ mà chỉ 1 phần ký tự đứng trước nó. Ví dụ: hello\B sẽ trùng khớp với chữ hello trong chuỗi helloworld nhưng sẽ không khớp với chuỗi hello world.
* \d Trùng khớp 1 ký tự số (digit).
* \D Trùng khớp 1ký tự không phải số (non-digit).
* \s Trùng khớp 1 ký tự khoảng trắng (whitespace) bao gồm khoảng trắng tạo ra bởi phím Tab.
* \S Trùng khớp với 1 ký tự không phải là khoảng trắng (non-whitespace).
* \w Trùng khớp với các ký tự là từ (word) bao gồm dấu \_ (underscore) và chữ số.
* \W Trùng khớp với các ký tự không phải là từ (non-word). Ví dụ: \W sẽ khớp với ký tự % trong chuỗi "100%".
* \uxxxx Trùng khớp với 1 ký tự unicode. Ví dụ: \u00FA sẽ khớp với ký tự "ú", \u00F9 sẽ khớp với ký tự "ù".
* \pL Trùng khớp với một ký tự Unicode bất kỳ ngoại trừ dấu cách. Đây chính là cú pháp viết hoàn hảo hơn của dấu .,Ví dụ \pL+ sẽ lấy được chuỗi truyền, thuyết trong chuỗi "truyền thuyết".

# Tóm lại

Regular expressions (hay còn gọi là Biểu thức chính quy – viết tắt là RegEx) là một chuỗi ký tự đặc biệt được dùng làm mẫu (pattern) để phân tích sự trùng khớp (match) của một tập hợp các chuỗi con cần lấy ra từ một chuỗi cha.

Chuỗi cha sau khi kiểm tra và lấy ra được các chuỗi con thì kết quả được đưa vào các Nhóm kết quả (Matches).

Trong mỗi Matches có các Group chứa các chuỗi con có thể cut lấy ra khỏi chuỗi cha (do người dùng định nghĩa chuỗi con cần lấy hoặc không lấy bằng cách viết thêm cặp ngoặc tròn () bao bọc chuỗi con cần lấy)

RegEx có thể lấy chuỗi ra (cut) hoặc thay thế chuỗi (replace)

Hầu hết những ngôn ngữ lập trình (PHP, C#, Perl, Javascript…) đều cung cấp các thư viện hoặc hàm xử lý để lập trình viên có thể làm việc với Regular expression.

Tham khảo

Tham khảo thêm toàn bộ cú pháp RegEx bằng tiếng Anh tại: https://www.cheatography.com/davechild/cheat-sheets/regular-expressions/

Một số công cụ test RegEx Online

https://regex101.com/

http://regexr.com/

http://www.regexpal.com/

<http://regexper.com/>

# Ví dụ đơn giản

## Tìm ngày tháng chính xác

/\b(0?[1-9]|[12]\d|3[01])[\/\-.](0?[1-9]|[12]\d|3[01])[\/\-.](\d{2}|\d{4})\b/g

Bắt các chuỗi ngày tháng này

12/01/2017, 15/2/2017, 3/03/2018, 12/24/2018, 1.1.2018, 7-9-18, 1.31.2018

Không bắt các chuỗi như thế này, vì không đúng ngày tháng

123/2/1234 13/ab/3346 12/223/12 60/60/2018 32/12/2018

Link demo: <https://regex101.com/r/3dNzjU/2>

## Check một chuỗi ngày tháng bất kỳ có hợp lệ hay không tính theo năm nhuận

Biểu thức RegEx:

^(((0?[1-9]|1[012])/(0?[1-9]|1\d|2[0-8])|(0?[13456789]|1[012])/(29|30)|(0?[13578]|1[02])/31)/(19|[2-9]\d)\d{2}|0?2/29/((19|[2-9]\d)(0[48]|[2468][048]|[13579][26])|(([2468][048]|[3579][26])00)))$

Sẽ đúng cho:

01/31/1905 | 1/9/1900 | 2/29/1904

Và sẽ không trùng với:

31/01/2005 | 02/29/2005 | 2/29/2005

## Check 1 chuỗi ngày tháng phải có cả ngày và giờ

Biểu thức RegEx:

^(?=\d)(?:(?:31(?!.(?:0?[2469]|11))|(?:30|29)(?!.0?2)|29(?=.0?2.(?:(?:(?:1[6-9]|[2-9]\d)?(?:0[48]|[2468][048]|[13579][26])|(?:(?:16|[2468][048]|[3579][26])00)))(?:\x20|$))|(?:2[0-8]|1\d|0?[1-9]))([-./])(?:1[012]|0?[1-9])\1(?:1[6-9]|[2-9]\d)?\d\d(?:(?=\x20\d)\x20|$))?(((0?[1-9]|1[012])(:[0-5]\d){0,2}(\x20[AP]M))|([01]\d|2[0-3])(:[0-5]\d){1,2})?$

Sẽ đúng cho:

31/12/2003 11:59:59 PM | 29-2-2004 | 01:45:02

Và sẽ không trùng với:

12/31/2003 | 29.02.2005 | 13:30 PM

## Check 1 chuỗi bất kỳ xem có phải ngày hoặc giờ hay không

Biểu thức RegEx:

^(?=\d)(?:(?!(?:(?:0?[5-9]|1[0-4])(?:\.|-|\/)10(?:\.|-|\/)(?:1582))|(?:(?:0?[3-9]|1[0-3])(?:\.|-|\/)0?9(?:\.|-|\/)(?:1752)))(31(?!(?:\.|-|\/)(?:0?[2469]|11))|30(?!(?:\.|-|\/)0?2)|(?:29(?:(?!(?:\.|-|\/)0?2(?:\.|-|\/))|(?=\D0?2\D(?:(?!000[04]|(?:(?:1[^0-6]|[2468][^048]|[3579][^26])00))(?:(?:(?:\d\d)(?:[02468][048]|[13579][26])(?!\x20BC))|(?:00(?:42|3[0369]|2[147]|1[258]|09)\x20BC))))))|2[0-8]|1\d|0?[1-9])([-.\/])(1[012]|(?:0?[1-9]))\2((?=(?:00(?:4[0-5]|[0-3]?\d)\x20BC)|(?:\d{4}(?:$|(?=\x20\d)\x20)))\d{4}(?:\x20BC)?)(?:$|(?=\x20\d)\x20))?((?:(?:0?[1-9]|1[012])(?::[0-5]\d){0,2}(?:\x20[aApP][mM]))|(?:[01]\d|2[0-3])(?::[0-5]\d){1,2})?$

Sẽ đúng cho:

31.12.6008 | 5:30 AM | 30-04-1066

Và sẽ không trùng với:

00/00/0000 | 99:99:99 | 29/02/2005

## Cách bắt lấy chuỗi bất kỳ

<label .\*?>(.\*?)<\/label>

Lấy được các thông tin giữa 2 thẻ label

<label class="s-nav right" for="s-22">simply</label>

<label class="s-nav left" for="s-21">dummy</label>

<label class="s-nav right" for="s-23">text</label>

<label class="s-nav left" for="s-22">of the printing</label>

<label class="s-nav right" for="s-24">and typesetting</label>

<label class="s-nav left" for="s-23">industry</label>

<label class="s-nav right" for="s-25">Lorem Ipsum</label>

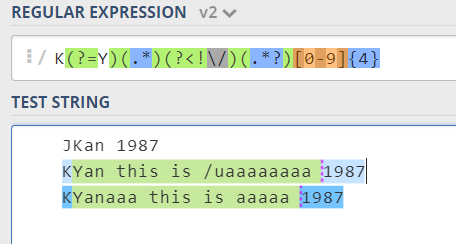
<label class="s-nav left" for="s-24">Ipsum</label>

Link demo: https://regex101.com/r/3dNzjU/3

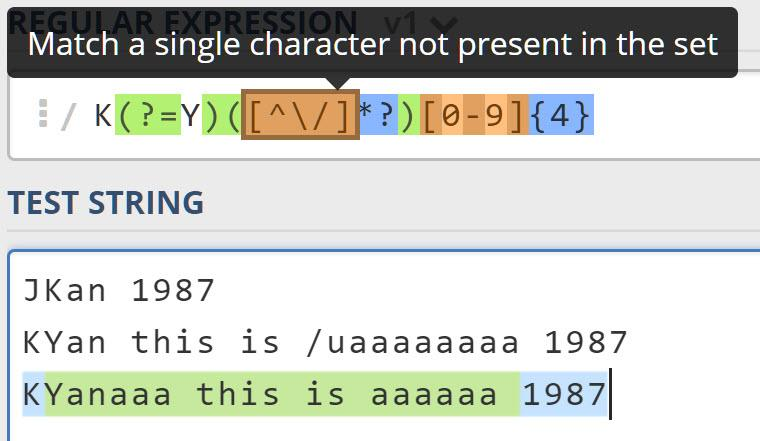
## Lấy một chuỗi nhưng không chứa 1 ký tự đặc biệt nào đó

/K(?=Y)([^\/]\*?)[0-9]{4}/g

Ví dụ bạn cần lấy 1 chuỗi dài, nhưng nếu chuỗi đó có chứa 1 ký tự đặt biệt nào đó, ví dụ ký tự / thì không được lấy dòng đó. Nghĩa là cần loại bỏ 1 ký tự đặc biệt ra khỏi tập kết quả (group matches) thu được.



Rất đơn giản, chúng ta hãy làm như sau để có thể loại bỏ dấu "/" ra khỏi kết quả thu được. https://regex101.com/r/ctuQwj/1



Giải thích 1 chút, như mình đã nói bên trên, khi dùng [^ab] thì nó sẽ loại bỏ các kết quả có chứa ký tự a hoặc b ra khỏi tập đã lấy được. Ở đây bạn cần loại bỏ dấu / nên ta sẽ viết là [^\/]. Và vì chúng ta cần nhân bản nhiều lần các ký tự khác dấu / để tìm 1 chuỗi dài liên tiếp, cho đến khi gặp 1 số có 4 chữ số. Vậy thì ta sẽ viết thêm đuôi \*? vào để thành [^\/]\*?.

Bình thường nếu tìm một chuỗi dài liên tiếp, không quan tâm là ký tự hay chữ số, ta hay viết là .\*?, do đó cách viết [^\/]\*? thực ra là một biến thể của .\*?.

Dấu chấm . sẽ bắt được một ký tự bất kỳ ngoại trừ dấu xuống dòng, do đó .\*? sẽ chỉ bắt được chuỗi dài vô tận ở cùng 1 dòng (1 line). Do đó nếu bạn cần bắt được chuỗi nằm ở nhiều dòng cho đến khi gặp một đoạn chuỗi cố định nào đó, thì sẽ phải dùng đến chiêu sau đây.

Sử dụng [\s\S] để thay thế cho dấu . ta sẽ bắt được toàn bộ ký tự bất kỳ bao gồm cả dấu xuống dòng /n. Do đó để bắt được 1 chuỗi dài vô tận nằm ở nhiều dòng ta sẽ dùng [\s\S]\*

## Lấy một chuỗi nhưng không chứa 1 chuỗi con đặc biệt nào đó

/(K((?!\/bc|\/a).)\*?)(\d{4})/g

Bây giờ bạn không chỉ cần exclude (loại bỏ) một ký tự ra khỏi danh sách chuỗi thu được, bạn còn cần loại bỏ hẳn "một chuỗi" thì sao. Vậy hãy dùng cách sau đây:

Vấn đề lúc này cần phải dùng đến chức năng Negative Lookahead của Regex, cách hoạt động như sau:



Giả sử mình cần exclude (loại bỏ) một chuỗi (chứ không phải 1 ký tự như là ví dụ [^\/]\*? như bên trên nữa) ra khỏi một chuỗi đã bắt được. Thì cần dùng cách viết (?!abc) (Chú ý là dấu ( và ) là bắt buộc phải viết để đủ cú pháp). Trong đó abc là 1 chuỗi các ký tự liền nhau cần loại bỏ. Nếu cần loại bỏ 2 hoặc nhiều chuỗi con thì sao? vậy ta sẽ viết là (?!abc|def|gho).

Vì chuỗi con nằm bên trong chuỗi cha, do đó chúng ta cần viết thêm 1 dấu chấm . ngay đằng sau (?!\/bc) và bọc toàn bộ khối đó lại thành 1 group. Rồi viết thêm \*? bên ngoài để nhân bản các ký tự lên thành 1 chuỗi.

Như vậy để viết Regex này ta phải tư duy từ bên trong ra bên ngoài. Và việc viết thêm Negative Lookahead kèm với dấu chấm dạng ((?!abc).)\*? thật ra cũng vẫn chính là một biến thể của .\*? (bắt một chuỗi dài vô tận, nhưng không được chứa chuỗi con cần loại bỏ bên trong nó)

Link demo: https://regex101.com/r/SThqo2/2

## Bắt 1 chuỗi bất kỳ nằm ở nhiều dòng (mutiline)

<h1>[\s\S]\*<\/h2>

Sẽ bắt được cả toàn bộ chuỗi nhiều dòng sau đây:

<h1>My First Heading</h1>

<p>My first paragraph.</p>

<h2>My first paragraph.</h2>

Giải thích tại sao lại bắt được. Rất đơn giản, là vì \s sẽ lấy được ký tự dấu cách, còn \S sẽ lấy được ký tự bất kỳ ngoại trừ dấu cách (và \S sẽ bắt được dấu xuống dòng \n). Nếu kết hợp 2 ký tự đó nhét vào một khung vuông [\s\S] tức là sẽ lấy được 1 trong 2 trường hợp. Sẽ bắt được toàn bộ các ký tự bất kỳ bao gồm cả dấu cách nếu gặp. Viết [\s\S]\*? thật ra là nhân bản ký tự cần bắt lên nhiều lần để tìm một chuỗi dài vô tận.

Bình thường nếu tìm một chuỗi dài liên tiếp, không quan tâm là ký tự hay chữ số, ta hay viết là .\*?, do đó cách viết [\s\S]\*? thực ra là một biến thể của .\*?. Dấu chấm . sẽ bắt được một ký tự bất kỳ ngoại trừ dấu xuống dòng, do đó .\*? sẽ chỉ bắt được chuỗi dài vô tận ở cùng 1 dòng (1 line). Do đó nếu bạn cần bắt được chuỗi nằm ở nhiều dòng cho đến khi gặp một đoạn chuỗi cố định nào đó, thì sẽ phải dùng đến chiêu sau đây. Sử dụng [\s\S] để thay thế cho dấu . ta sẽ bắt được toàn bộ ký tự bất kỳ bao gồm cả dấu xuống dòng /n. Do đó để bắt được 1 chuỗi dài vô tận nằm ở nhiều dòng ta sẽ dùng [\s\S]\*

Các bạn có thể xem link demo: https://regex101.com/r/irhKCk/1

## BTU1. Tìm ra các chuỗi là số điện thoại trong chuỗi sau.

Chú ý một số loại ra một số chuỗi không phải là số điện thoại

^\s\*(?:\+?(\d{1,3}))?([-. (]\*(\d{3})[-. )]\*)?((\d{3})[-. ]\*(\d{2,4})(?:[-.x ]\*(\d+))?)\s\*$

* Option cho chuỗi regex phải là /gm

Positive:

+42 555.123.4567 +1-(800)-123-4567 +7 555 1234567

+7(926)1234567

(926) 1234567

+79261234567

926 1234567

9261234567

1234567

123-4567

123-89-01

495 1234567

469 123 45 67

89261234567

8 (926) 1234567

926.123.4567

415-555-1234

650-555-2345

(416)555-3456

202 555 4567

4035555678

1 416 555 9292

11234567890

123-456-7890

(123) 456-7890

1 (123) 456-7890

123 456 7890

1 123 456 7890

123.456.7890

1 123.456.7890

1.123.456.7890

+1 123.456.7890

+91 (123) 456-7890

+1-(800)-555-2468

1-234-567-8912

+22-432-359-3687

Negative:

926 3 4

8 800 600-APPLE

Lời giải <https://regexr.com/38pvb>

## Xác thực số điện thoại

^\+?\d{1,3}?[- .]?\(?(?:\d{2,3})\)?[- .]?\d\d\d[- .]?\d\d\d\d$

* Option cho chuỗi regex phải là /gm

Ngắn gọn, ngọt ngào và đi thẳng vào vấn đề. Đoạn regex này sẽ xác thực bất cứ cú pháp số điện thoại nào dựa trên phong cách số điện thoại Mỹ.

## BTU2. Tìm ra chuỗi là địa chỉ email trong chuỗi sau

Chú ý loại bỏ các chuỗi k phải địa chỉ emai thực sự.

^([A-Z|a-z|0-9](\.|\_){0,1})+[A-Z|a-z|0-9]\@([A-Z|a-z|0-9])+((\.){0,1}[A-Z|a-z|0-9]){2}\.[a-z]{2,3}$

Acceptable

---------------

abc1.1@domain.com

abc.abc@domain.com

abc\_41.15768@domain.com

12345@domain.com

12.12.23@domain.com

ABC.ABC.ABC.ABC0046@domain.com

abc\_123\_4595@domain.com

abc@asdas.co.in

abc@insta123.com

abc@inta.co.in

Non-Acceptable

---------------

abc@@insta.com.com.com

abc@insta..com

abc@insta%.com

abc@insta/.com

abc@insta\\.com

abc..abc@domain.com

abc\_\_abc@domain.com

abc.@insta.com

abc..de@insta.com

abc-@insta.com

abc-def@insta.com

abc+@insta.com

abc123..45@insta.com

abc%@insta.com

abc\\@insta.com

abc/@insta.com

abc@insta-.com

ABC.ABC.ABC.ABC0046@domain.com.com.com

Lời giải: <https://regexr.com/3bcrb>

## Khớp một Email

/^([a-z0-9\_\.-]+)@([\da-z\.-]+)\.([a-z\.]{2,6})$/

Mô tả:

Chúng ta bắt đầu bằng cách nói cho trình phân tích cú pháp tìm đầu chuỗi (^). Trong nhóm đầu tiên, chúng ta khớp một hoặc nhiều chữ cái thường, số, dấu gạch dưới, dấu chấm hoặc dấu gạch ngang. Tôi đã thoát dấu chấm vì một dấu chấm không thoát có nghĩa là bất kỳ ký tự nào. Ngay sau đó, phải có một ký hiệu. Tiếp theo tên miền phải là: một hoặc nhiều chữ cái thường, số, dấu gạch dưới, dấu chấm hoặc dấu gạch nối. Sau đó, một dấu chấm khác (đã thoát), với phần mở rộng là từ hai đến sáu chữ cái hoặc dấu chấm. Tôi có 2 đến 6 bởi vì TLD cụ thể của quốc gia (.ny.us hoặc .co.uk). Cuối cùng, chúng ta kết thúc chuỗi ($).

Chuỗi khớp:

john@doe.com

Chuỗi không khớp:

john@doe.something (TLD quá dài)

## BTU3. Tìm cách lấy ra và ngăn cách chuỗi số sau thành có phần trăm phần nghìn.

\d{1,3}(?=(\d{3})+(?!\d))

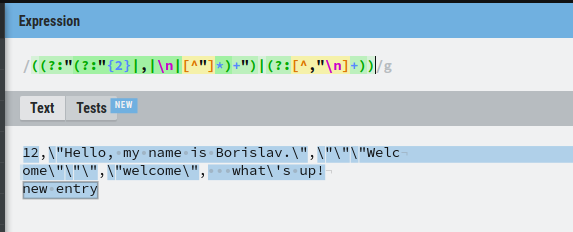


Lời giải: regexr.com/4409r

## BTU4. Lấy ra các chuỗi ngăn cách bởi dấu phẩy theo định dạng của một file CSV như sau

chú ý là chuỗi bị xuống dòng:

((?:"(?:"{2}|,|\n|[^"]\*)+")|(?:[^,"\n]+))



Lời giải: https://regexr.com/3apuc

## BTU5. Tìm và xóa đi toàn bộ các thẻ HTML trong đoạn text dưới đây:

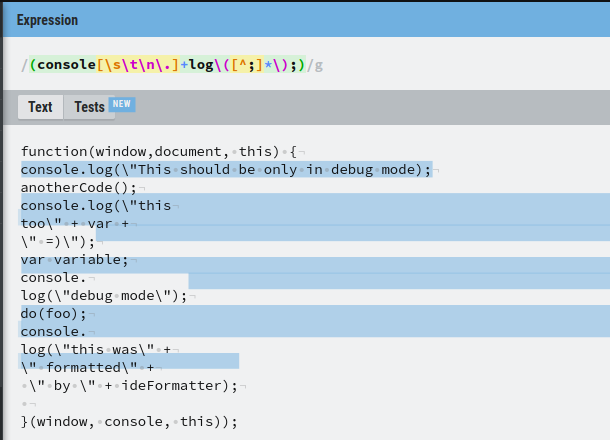
<.\*?>



Lời giải: https://regexr.com/3cak1

## BTU6. Tìm cách loại bỏ các chuỗi console.log() do lập trình viên viết để test ra khỏi code trước khi bàn giao

(console[\s\t\n\.]+log\([^;]\*\);)



Lời giải: https://regexr.com/3fi66

## BTU7. Loại bỏ toàn bộ các đoạn comment code trong đoạn code sau đây:

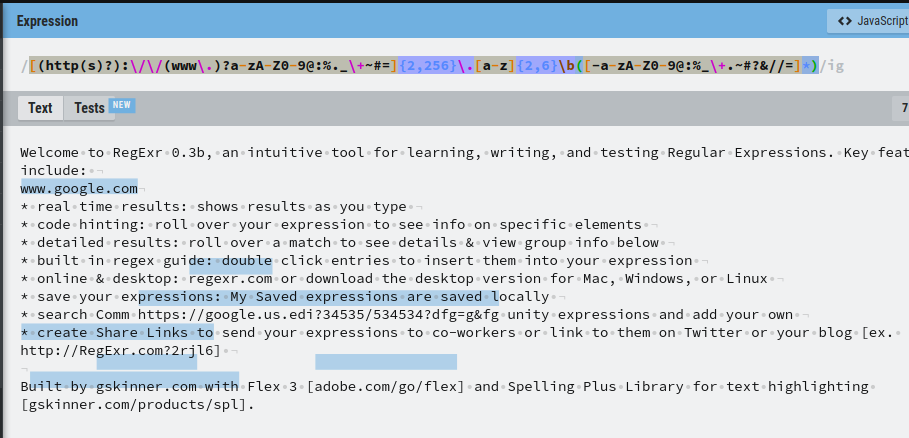
\/\\*[\s\S]\*?\\*\/|\/\/.\*



Lời giải: https://regexr.com/3aeb7

## BTU8. Lấy ra các chuỗi là các URL đầy đủ trong đoạn text dưới đây:

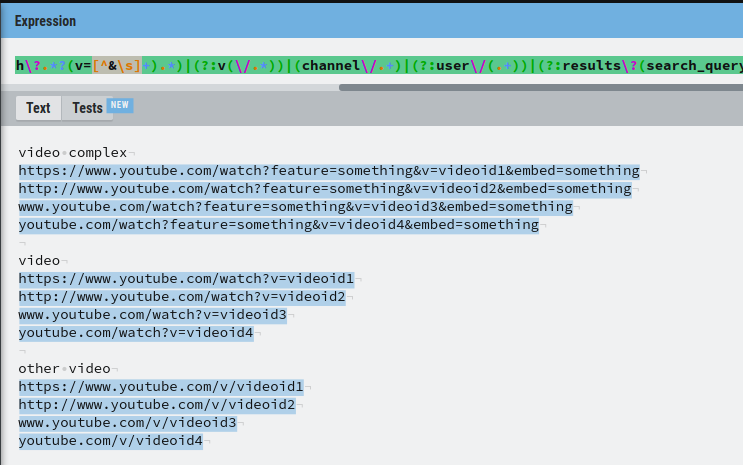
[(http(s)?):\/\/(www\.)?a-zA-Z0-9@:%.\_\+~#=]{2,256}\.[a-z]{2,6}\b([-a-zA-Z0-9@:%\_\+.~#?&//=]\*)



Lời giải: https://regexr.com/39nr7

## BTU9. Lấy ra toàn bộ các URL của một video Youtube từ đoạn text test sau đây

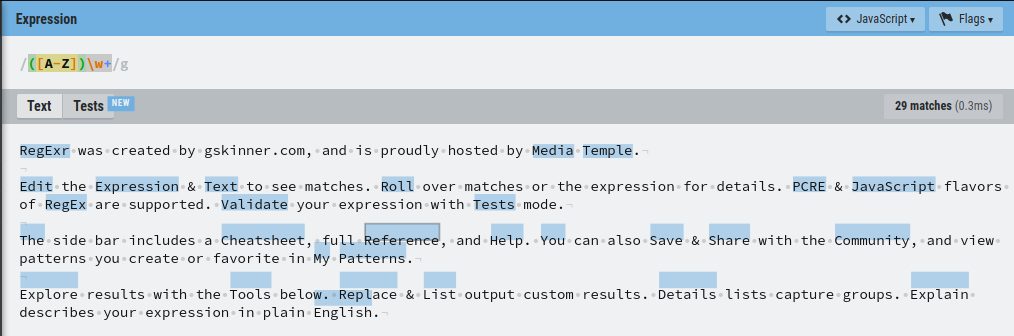
(?:https?:\/\/)?(?:(?:(?:www\.?)?youtube\.com(?:\/(?:(?:watch\?.\*?(v=[^&\s]+).\*)|(?:v(\/.\*))|(channel\/.+)|(?:user\/(.+))|(?:results\?(search\_query=.+))))?)|(?:youtu\.be(\/.\*)?))



Lời giải: https://regexr.com/3akf5

## BTU10. Lấy ra các đoạn text là giá trị màu RBG

([A-Z])\w+



Lời giải: <https://regexr.com/38lmo>

## Độ mạnh của mật khẩu

^(?=.\*[A-Z].\*[A-Z])(?=.\*[!@#$&\*])(?=.\*[0-9].\*[0-9])(?=.\*[a-z].\*[a-z].\*[a-z]).{8}$

Kiểm tra độ mạnh của một mật khẩu thường là chủ quan nên không có câu trả lời chính xác tuyệt đối. Nhưng tôi cảm thấy đoạn regex này là một điểm khởi đầu tuyệt vời nếu bạn không muốn phải viết riêng hàm kiểm tra độ mạnh mật khẩu của bạn từ đầu.

## Khớp một Mật khẩu

Matching a password

/^[a-z0-9\_-]{6,18}$/

Mô tả:

Khớp một mật khẩu rất giống với việc khớp một tên người dùng. Sự khác biệt duy nhất là thay vì 3 đến 16 chữ cái, các con số, dấu gạch dưới hoặc dấu gạch ngang, thì chúng ta muốn 6 đến 18 trong số đó ({6,18}).

Chuỗi khớp:

myp4ssw0rd

Chuỗi không khớp:

mypa$$w0rd (chứa một dấu đô la)

## Mã màu Hex

\#([a-fA-F]|[0-9]){3, 6}

Mã màu hex rất phố biến trong lĩnh vực phát triển web. Đoạn regex này có thể được sử dụng để lấy mã hex phù hợp từ chuỗi bất kỳ cho bất cứ mục đích nào.

## Địa chỉ IPv4

/\b(?:(?: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]?)\b/

Tương tự như một địa chỉ email là địa chỉ IP - được sử dụng để xác định một máy tính cụ thể truy cập Internet. Biểu thức chính quy này sẽ kiểm tra một chuỗi xem nó có tuân theo cú pháp địa chỉ IPv4 hay không.

## Khớp một Địa chỉ IP

/^(?:(?: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]?)$/

Mô tả:

Bây giờ, tôi sẽ không nói dối, không phải tôi viết regex này; Tôi lấy nó từ đây. Bây giờ, điều đó không có nghĩa là tôi không thể phân tích nó.

Nhóm đầu tiên thực sự không phải là một nhóm khớp bởi vì

1

được đặt bên trong cái sẽ nói với trình phân tích cú pháp không khớp nhóm này (điều này sẽ rõ hơn trong regex cuối cùng). Chúng ta cũng muốn nhóm không khớp này được lặp lại ba lần — {3} ở cuối nhóm. Nhóm này chứa một nhóm khác, một nhóm con và một dấu chấm. Trình phân tích cú pháp tìm kiếm một khớp trong nhóm con sau đó một dấu chấm để tiếp tục.

Nhóm con cũng là một nhóm không khớp khác. Nó chỉ là một nhóm các bộ ký tự (những thứ bên trong ngoặc đơn): chuỗi "25" theo sau bởi một số từ 0 đến 5; hoặc chuỗi "2" và một số từ 0 đến 4 và bất kỳ số nào; hoặc một tuỳ chọn số không hoặc một theo sau bởi hai số, với cái thứ hai là tùy chọn.

Sau khi chúng ta khớp ba trong số đó, nó sẽ vào nhóm không khớp tiếp theo. Điều này muốn: chuỗi "25" theo sau bởi một số từ 0 đến 5; hoặc chuỗi "2" với một số từ 0 đến 4 và một số khác ở cuối; hoặc một tuỳ chọn số không hoặc một theo sau bởi hai số, với cái thứ hai là tùy chọn.

Chúng ta kết thúc regex khó hiểu này bằng chuỗi kết thúc.

Chuỗi khớp:

73.60.124.136 (không, đó không phải địa chỉ IP của tôi :P)

Chuỗi không khớp:

256.60.124.136 (nhóm đầu tiên phải là "25" và một số từ 0 đến 5)

## Địa chỉ IPv6

(([0-9a-fA-F]{1,4}:){7,7}[0-9a-fA-F]{1,4}|([0-9a-fA-F]{1,4}:){1,7}:|([0-9a-fA-F]{1,4}:){1,6}:[0-9a-fA-F]{1,4}|([0-9a-fA-F]{1,4}:){1,5}(:[0-9a-fA-F]{1,4}){1,2}|([0-9a-fA-F]{1,4}:){1,4}(:[0-9a-fA-F]{1,4}){1,3}|([0-9a-fA-F]{1,4}:){1,3}(:[0-9a-fA-F]{1,4}){1,4}|([0-9a-fA-F]{1,4}:){1,2}(:[0-9a-fA-F]{1,4}){1,5}|[0-9a-fA-F]{1,4}:((:[0-9a-fA-F]{1,4}){1,6})|:((:[0-9a-fA-F]{1,4}){1,7}|:)|fe80:(:[0-9a-fA-F]{0,4}){0,4}%[0-9a-zA-Z]{1,}|::(ffff(:0{1,4}){0,1}:){0,1}((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9])\.){3,3}(25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9])|([0-9a-fA-F]{1,4}:){1,4}:((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9])\.){3,3}(25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9]))

Hoặc có thể bạn sẽ muốn kiểm tra một địa chỉ theo cú pháp IPv6 mới hơn với đoạn regex nâng cao hơn này. Sự khác biệt là rất nhỏ mặc dù nó quan trọng trong quá trình phát triển.

## Dấu phân cách hàng nghìn

/\d{1,3}(?=(\d{3})+(?!\d))/g

Mã regex này hoạt động trên bất kỳ số nào và sẽ áp dụng bất cứ dấu phân cách nào bạn chọn cho mỗi chữ số thứ ba phân tách thành hàng ngàn, hàng triệu,...

## Thêm HTTP vào trước liên kết

if (!s.match(/^[a-zA-Z]+:\/\//))

{

s = 'http://' + s;

}

Cho dù bạn đang làm việc trong JavaScript, Ruby hay PHP, biểu thức này có thể tỏ ra rất hữu ích. Nó sẽ kiểm tra bất kỳ chuỗi URL nào để xem nếu nó có tiền tố HTTP/HTTPS hay không, và nếu không, thêm vào trước chuỗi đó cho phù hợp.

## Lấy tên miền từ URL

/https?:\/\/(?:[-\w]+\.)?([-\w]+)\.\w+(?:\.\w+)?\/?.\*/i

Mỗi tên miền trang web đều chứa giao thức ở đầu (HTTP hoặc HTTPS) và đôi khi có tên miền phụ với đường dẫn trang bổ sung. Bạn có thể sử dụng đoạn regex này để cắt qua tất cả những điều đó và chỉ trả lại tên miền.

## Sắp xếp các từ khóa bằng cách đếm số từ

^[^\s]\*$ matches exactly 1-word keyword

^[^\s]\*\s[^\s]\*$ matches exactly 2-word keyword

^[^\s]\*\s[^\s]\* matches keywords of at least 2 words (2 and more)

^([^\s]\*\s){2}[^\s]\*$ matches exactly 3-word keyword

^([^\s]\*\s){4}[^\s]\*$ matches 5-words-and-more keywords (longtail)

Những ai dùng Google Analytics và Webmaster Tools sẽ thực sự thích biểu thức chính quy này. Nó có thể sắp xếp các từ khoá dựa trên số các từ được sử dụng trong một tìm kiếm.

Đây có thể là số lượng cụ thể (tức là chỉ có 5 từ) hoặc nó có thể phù hợp với một loạt các từ (tức là 2 hoặc nhiều hơn). Khi được sử dụng để sắp xếp và phân tích dữ liệu, nó là một trong những biểu thức chính quy mạnh mẽ.

## Tìm một chuỗi Base64 hợp lệ trong PHP

\?php[ \t]eval\(base64\_decode\(\'(([A-Za-z0-9+/]{4})\*([A-Za-z0-9+/]{3}=|[A-Za-z0-9+/]{2}==)?){1}\'\)\)\;

Nếu bạn là một nhà phát triển PHP thì đôi khi bạn có thể cần phân tích qua code để tìm kiếm các đối tượng nhị phân đã mã hóa Base64 (ví dụ như tìm shell được giấu trong một tập tin PHP chẳng hạn). Đoạn regex này có thể áp dụng với tất cả code PHP và sẽ kiểm tra bất cứ chuỗi Base64 nào đang tồn tại.

## Khoảng trắng ở đầu và cuối

^[ \s]+|[ \s]+$

Sử dụng đoạn mã này để lấy ra khoảng trắng ở đầu/cuối từ một chuỗi. Đây có thể không phải là một việc lớn, nhưng đôi khi nó ảnh hưởng đến đầu ra khi lấy kết quả ra từ một cơ sở dữ liệu hoặc áp dụng vào mã hóa tài liệu.

## Lấy nguồn ảnh

\< \*[img][^\>]\*[src] \*= \*[\"\']{0,1}([^\"\'\ >]\*)

Nếu vì một số lý do nào đó mà bạn cần lấy ra một nguồn ảnh trực tiếp từ HTML thì đoạn mã này là giải pháp hoàn hảo. Mặc dù nó có thể chạy trơn tru trên Backend, các nhà phát triển JS nên dựa vào phương thức .attr() của jQuery cho Frontend.

## Xác thực ngày trong định dạng DD/MM/YYYY

^(?:(?:31(\/|-|\.)(?:0?[13578]|1[02]))\1|(?:(?:29|30)(\/|-|\.)(?:0?[1,3-9]|1[0-2])\2))(?:(?:1[6-9]|[2-9]\d)?\d{2})$|^(?:29(\/|-|\.)0?2\3(?:(?:(?:1[6-9]|[2-9]\d)?(?:0[48]|[2468][048]|[13579][26])|(?:(?:16|[2468][048]|[3579][26])00))))$|^(?:0?[1-9]|1\d|2[0-8])(\/|-|\.)(?:(?:0?[1-9])|(?:1[0-2]))\4(?:(?:1[6-9]|[2-9]\d)?\d{2})$

Các ngày khó khăn vì chúng có thể xuất hiện dưới dạng văn bản + số, hoặc chỉ là số với các định dạng khác nhau. PHP có một hàm ngày tuyệt vời nhưng điều này không phải luôn là lựa chọn tốt nhất khi lấy ra từ một chuỗi thô (raw). Cân nhắc việc thay thế bằng cách sử dụng biểu thức chính quy này cho các cú pháp ngày cụ thể.

## Khớp ID Video Youtube

/https?:\/\/(?:youtu\.be\/|(?:[a-z]{2,3}\.)?youtube\.com\/watch(?:\?|#\!)v=)([\w-]{11}).\*/gi

Youtube đã giữ cấu trúc URL giống nhau trong nhiều năm chỉ bởi nó hoạt động. Đây cũng là trang chia sẻ video phổ biến nhất trên web, vì thế các video Youtube có xu hướng điều phối lưu lượng truy cập nhiều nhất.

Nếu bạn cần lấy ra một ID video Youtube từ một URL thì đoạn regex này là hoàn hảo và hoạt động tốt với tất cả các biến thể của cấu trúc URL Youtube.

## Xác thực ISBN

/\b(?:ISBN(?:: ?| ))?((?:97[89])?\d{9}[\dx])\b/i

Sách được in theo một hệ thống số gọi là ISBN. Điều này có thể lấy khá khó khăn khi bạn xem xét sự khác biệt giữa ISBN-10 và ISBN-13.

Tuy nhiên đoạn regex đáng kinh ngạc này cho phép bạn để xác nhận một số ISBN và kiểm tra xem nó là ISBN10 hay 13. Tất cả các mã được viết bằng PHP vì vậy điều này có thể tỏ ra đặc biệt hữu ích cho các nhà phát triển web.

## Kiểm tra mã bưu điện

^\d{5}(?:[-\s]\d{4})?$

Tác giả của đoạn regex này không chỉ chia sẻ nó miễn phí mà còn dành thời gian để giải thích nó. Bạn sẽ thấy nó hữu ích cho dù bạn đang khớp một mã bưu điện loại 5 chữ số hay phiên bản dài 9 chữ số dài hơn.

Hãy nhớ rằng nó chủ yếu dành cho hệ thống mã bưu điện của Mỹ nên có thể bạn sẽ cần điều chỉnh cho các quốc gia khác.

## Tên người dùng Twitter hợp lệ

/@([A-Za-z0-9\_]{1,15})/

Đây là một đoạn mã regex rất nhỏ để khớp tên người dùng Twitter tìm thấy trong một chuỗi. Nó kiểm tra dựa trên cú pháp đề cập (@mention).

## Số thẻ tín dụng

^(?:4[0-9]{12}(?:[0-9]{3})?|5[1-5][0-9]{14}|6(?:011|5[0-9][0-9])[0-9]{12}|3[47][0-9]{13}|3(?:0[0-5]|[68][0-9])[0-9]{11}|(?:2131|1800|35\d{3})\d{11})$

Chứng thực số thẻ tín dụng thường bắt buộc một nền tảng máy chủ an toàn. Nhưng regex có thể được sử dụng cho các yêu cầu tối thiểu của một số thẻ tín dụng điển hình.

## Tìm thuộc tính CSS

^\s\*[a-zA-Z\-]+\s\*[:]{1}\s[a-zA-Z0-9\s.#]+[;]{1}

Có thể rất hiếm để chạy regex trên CSS nhưng nó không phải là một tình huống lạ.

Đoạn mã này có thể được sử dụng để lấy ra mọi thuộc tính và giá trị CSS khớp từ selector cụ thể. Nó có thể được sử dụng cho bất kỳ lý do nào, ví dụ như loại bỏ thuộc tính trùng lặp.

## Loại bỏ chú thích HTML

<!--(.\*?)-->

Nếu vì bất cứ lý do nào mà bạn cần phải loại bỏ tất cả các chú thích từ một khối HTML thì đây là mã regex cần dùng. Bạn sẽ tìm thấy một ví dụ PHP sử dụng preg\_replace.

## URL trang cá nhân Facebook

/(?:http:\/\/)?(?:www\.)?facebook\.com\/(?:(?:\w)\*#!\/)?(?:pages\/)?(?:[\w\-]\*\/)\*([\w\-]\*)/

Facebook rất phổ biến và có nhiều sự sắp đặt URL khác nhau. Trong một tình huống mà bạn đang lấy URL hồ sơ từ người sử dụng thì đoạn regex này có thể hữu ích để phân tích chuỗi và xác nhận rằng chúng có cấu trúc đúng. Đoạn này có thể làm chính xác điều đó và nó hoàn hảo cho tất cả các kiểu liên kết FB.

## Kiểm tra phiên bản Internet Explorer

^.\*MSIE [5-8](?:\.[0-9]+)?(?!.\*Trident\/[5-9]\.0).\*$

Trình duyệt IE vẫn đang có thị phần người dùng lớn và các nhà phát triển thường phải kiểm tra phiên bản IE để xử lý tính tương thích của trang web.

Đoạn này có thể được sử dụng trong JavaScript để kiểm tra phiên bản của Internet Explorer (5-11) đang được sử dụng dựa trên chuỗi User-Agent.

## Bóc tách giá

/(\$[0-9,]+(\.[0-9]{2})?)/

Giá cả đi kèm trong một loạt các định dạng có chứa số thập phân, dấu phẩy và ký hiệu tiền tệ. Biểu thức này có thể kiểm tra tất cả các định dạng khác nhau để lấy ra một mức giá từ bất kỳ chuỗi nào.

## Phân tích tiêu đề Email

/\b[A-Z0-9.\_%+-]+@(?:[A-Z0-9-]+\.)+[A-Z]{2,6}\b/i

Với một dòng code này bạn có thể phân tích thông qua một tiêu đề email để lấy ra thông tin "to" từ tiêu đề. Nó có thể được sử dụng song song với nhiều email liên kết với nhau.

## Khớp một loại tập tin cụ thể

/^(.\*\.(?!(htm|html|class|js)$))?[^.]\*$/i

Khi bạn đang làm việc với các định dạng tập tin khác nhau như .xml, .html và .js, nó có thể giúp kiểm tra các tập tin cả nội bộ (local) và được tải lên bởi người dùng. Đoạn này lấy ra một phần mở rộng tập tin để kiểm tra xem nó có giá trị nằm trong một danh sách các phần mở rộng hợp lệ hay không.

## Khớp một chuỗi URL

/[-a-zA-Z0-9@:%\_\+.~#?&//=]{2,256}\.[a-z]{2,4}\b(\/[-a-zA-Z0-9@:%\_\+.~#?&//=]\*)?/gi

Đoạn này có thể sử dụng cho cả chuỗi HTTP và HTTPS để kiểm tra xem nó có khớp với cú pháp tên miền mức cao nhất (TLD) hay không. Cũng có một phiên bản đơn giản hơn đoạn này sử dụng RegExp của JavaScript.

## Thêm rel="nofollow" vào các liên kết

(<a\s\*(?!.\*\brel=)[^>]\*)(href="https?://)((?!(?:(?:www\.)?'.implode('|(?:www\.)?', $follow\_list).'))[^"]+)"((?!.\*\brel=)[^>]\*)(?:[^>]\*)>

Nếu bạn đang làm việc với hàng loạt mã HTML có thể nó thật khủng khiếp khi phải làm thủ công các nhiệm vụ lặp đi lặp lại. Biểu thức chính quy là hoàn hảo cho công việc này và họ sẽ tiết kiệm được rất nhiều thời gian.

Đoạn regex này có thể lấy ra tất cả liên kết từ một khối HTML và thêm thuộc tính rel="nofollow" vào mỗi phần tử.

## Khớp Media Query

/@media([^{]+)\{([\s\S]+?})\s\*}/g

Chia tách các Media Query CSS vào từng tham số và thuộc tính của chúng. Điều này có thể giúp bạn phân tích CSS bên ngoài với sự tập trung trực tiếp hơn vào cách mà code hoạt động.

## Cú pháp tìm kiếm Google

/([+-]?(?:'.+?'|".+?"|[^+\- ]{1}[^ ]\*))/g

Bạn có thể xây dựng mã regex của riêng bạn cho các thao tác tìm kiếm nội dung bằng cách sử dụng cú pháp độc quyền của Google. Các dấu cộng (+) biểu thị các từ khóa bổ sung và dấu trừ (-) biểu thị rằng nên bỏ qua và loại bỏ khỏi kết quả.

Đó là một đoạn khá phức tạp nhưng được sử dụng đúng cách thì nó có thể cung cấp một cơ sở cho việc xây dựng thuật toán tìm kiếm của riêng bạn.

## Khớp một Tên người dùng

Matching a username

/^[a-z0-9\_-]{3,16}$/

Mô tả:

Chúng ta bắt đầu bằng cách nói với trình phân tích cú pháp tìm đầu chuỗi (^), tiếp theo là bất kỳ chữ cái thường (a-z), con số (0-9), một dấu gạch dưới, hoặc một dấu gạch nối. Tiếp theo, {3,16} đảm bảo có ít nhất 3 trong số các ký tự đó, nhưng không quá 16. Cuối cùng, chúng ta kết thúc chuỗi ($).

Chuỗi khớp:

my-us3r\_n4m3

Chuỗi không khớp:

th1s1s-wayt00\_l0ngt0beausername (quá dài)

## Khớp một Giá trị Hex

/^#?([a-f0-9]{6}|[a-f0-9]{3})$/

Mô tả:

Chúng ta bắt đầu bằng cách nói cho trình phân tích cú pháp tìm đầu chuỗi (^). Tiếp theo, một ký hiệu con số là tùy chọn bởi vì nó được theo sau một dấu chấm hỏi. Dấu chấm hỏi nói cho trình phân tích cú pháp rằng ký tự trước — trong trường hợp này một ký hiệu số — là tùy chọn, nhưng phải nắm bắt nó nếu nó ở đó. Tiếp theo, bên trong nhóm đầu tiên (nhóm dấu ngoặc đơn đầu tiên), chúng ta có thể có hai tình huống khác nhau. Đầu tiên là bất kỳ chữ cái cái thường giữa a và f hoặc một con số sáu lần. Dấu gạch đứng cho chúng ta biết rằng chúng ta cũng có thể có ba chữ cái viết thường giữa a và f hoặc con số. Cuối cùng, chúng ta kết thúc chuỗi ($).

Lý do tôi đặt sáu ký tự phía trước đó là trình phân tích cú pháp sẽ bắt một giá trị hex giống như #ffffff. Nếu tôi đảo ngược nó để ba ký tự ở phía trước, thì trình phân tích cú pháp sẽ chỉ lấy #fff và không phải là ba f khác.

Chuỗi khớp:

#a3c113

Chuỗi không khớp:

#4d82h4 (chứa chữ h)

## Khớp một Slug

/^[a-z0-9-]+$/

Mô tả:

Bạn sẽ sử dụng regex này nếu bạn đã từng làm việc với mod\_rewrite và pretty URL. Chúng ta bắt đầu bằng cách nói cho trình phân tích cú pháp tìm đầu chuỗi (^), tiếp theo là một hoặc nhiều ký tự (dấu +), số hoặc dấu gạch ngang. Cuối cùng, chúng ta kết thúc chuỗi ($).

Chuỗi khớp:

my-title-here

Chuỗi không khớp:

my\_title\_here (chứa dấu gạch dưới)

## Khớp một URL

/^(https?:\/\/)?([\da-z\.-]+)\.([a-z\.]{2,6})([\/\w \.-]\*)\*\/?$/

Mô tả:

Regex này gần giống như lấy phần kết thúc của regex ở trên, ném nó vào giữa "http://" và một số cấu trúc tập tin ở cuối. Nghe có vẻ đơn giản hơn nhiều so với thực tế. Để bắt đầu, chúng ta tìm kiếm đầu dòng với dấu mũ (^).

Nhóm đầu tiên thì hoàn toàn tùy chọn. Nó cho phép URL bắt đầu với "http://", "https://", hoặc không phải cả hai. Tôi có một dấu chấm hỏi phía sau s để cho phép URL có http hoặc https. Để làm cho toàn bộ nhóm này là tùy chọn, tôi chỉ cần thêm một dấu chấm hỏi vào cuối của nó.

Tiếp theo là tên miền: một hoặc nhiều số, chữ cái, dấu chấm, hoặc dấu gạch ngang theo sau bởi một dấu chấm khác sau đó từ hai đến sáu chữ cái hoặc dấu chấm. Phần theo sau là các tập tin và thư mục tùy chọn. Trong nhóm, chúng ta muốn khớp bất kỳ số lượng dấu gạch chéo, ký tự, số, dấu gạch dưới, dấu cách, dấu chấm hoặc dấu gạch nối. Sau đó, chúng ta nói rằng nhóm này có thể được kết hợp nhiều lần khi chúng ta muốn. Điều này cho phép nhiều thư mục được khớp cùng với một tập tin ở cuối. Tôi đã sử dụng dấu sao thay vì dấu chấm hỏi bởi vì dấu sao có nghĩa là không hoặc nhiều, không phải không hoặc một. Nếu một dấu chấm hỏi đã được sử dụng ở đó, chỉ có một tập tin/thư mục sẽ có thể được khớp.

Sau đó, một dấu gạch chéo được khớp, nhưng nó có thể là tùy chọn. Cuối cùng chúng ta kết thúc với dấu kết thúc dòng.

Chuỗi khớp:

https://net.tutsplus.com/about

Chuỗi không khớp:

http://google.com/some/file!.html (chứa một dấu chấm than)

## Khớp một Thẻ HTML

/^<([a-z]+)([^<]+)\*(?:>(.\*)<\/\1>|\s+\/>)$/

Mô tả:

Một trong những regex hữu ích trong danh sách. Nó khớp bất kỳ thẻ HTML nào cùng với nội dung bên trong. Như thường lệ, chúng ta bắt đầu bằng đầu dòng.

Đầu tiên là tên của thẻ. Nó phải dài một hoặc nhiều chữ cái. Đây là nhóm đầu tiên, nó hữu ích khi chúng ta phải lấy thẻ đóng. Điều tiếp theo là các thuộc tính của thẻ. Đây là ký tự bất kỳ nhưng không phải dấu lớn hơn (>). Vì đây là tùy chọn, nhưng tôi muốn khớp nhiều hơn một ký tự, nên dấu sao được sử dụng. Dấu cộng tạo nên thuộc tính và giá trị, và dấu sao nói lên rằng có nhiều thuộc tính.

Tiếp theo là nhóm không khớp thứ ba. Bên trong, nó sẽ chứa một dấu lớn hơn, một số nội dung, và một thẻ đóng; hoặc một số khoảng trắng, một dấu gạch chéo, và dấu lớn hơn. Tùy chọn đầu tiên sẽ tìm kiếm một dấu lớn hơn theo sau bởi bất kỳ số ký tự nào, và thẻ đóng. \1 được sử dụng để đại diện cho nội dung được khớp trong nhóm đầu tiên. Trong trường hợp này nó là tên của thẻ. Bây giờ, nếu điều đó không thể được khớp thì chúng ta cần tìm một thẻ tự đóng (như thẻ img, br hoặc hr). Điều này cần có một hoặc nhiều khoảng trắng theo sau bởi "/>".

regex được kết thúc với dấu kết thúc dòng.

Chuỗi khớp:

Nettuts">http://net.tutsplus.com/">Nettuts+

Chuỗi không khớp:

<img src="img.jpg" alt="My image>" /> (các thuộc tính không thể chứa dấu lớn hơn)