

Регексите са регулярни изрази, които са като шаблони за търсене на някакъв определен текст, в друг такъв.

Java Advanced - Regular Expressions - януари 2017 - Петър Пенев

Regular Expressions

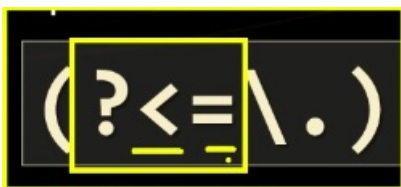
SOFTWARE UNIVERSITY FOUNDATION

- Sequence of characters that forms a search **pattern**

`(?<=\.){2,}(?=[A-Z])`

- Used for **finding and matching** certain parts of strings

I watch three climb before it's my turn. It's a tough one. The guy before me tries twice. He falls twice. After the last one, he comes down. He's finished for the day. It's my turn. My buddy says "good luck!" to me. I noticed a bit of a problem. There's an outcrop on this one. It's about halfway up the wall. It's not a



На горната снимка регекса означава, че преди този шаблон да намерим, трябва да има 1 точка. (Трябва да търсим нещо, което преди себе си или преди този шаблон, трябва да има една точка. Наклонената черта просто ескейпа точката, за да стане литерал иначе ще шаблона ще махва всичко.



След това казваме, че искаме да имаме 1 space , който се повтаря 2 или повече пъти.



Тук казваме, че след този шаблон(pattern), който съм намерил, искам да намеря някаква главна буква.

Резултатът от действието на този шаблон се вижда в следващата картинка.

I watch three climb before it's my turn. It's a tough one. The guy before me tries twice. He falls twice. After the last one, he comes down. He's finished for the day. It's my turn. My buddy says "good luck!" to me. I noticed a bit of a problem. There's an outcrop on this one. It's about halfway up the wall. It's not a

Тук ясно се вижда как шаблона хваща точката, двата или повече space-a и след това веднага има главна буква.

Следващата конструкция се нарича Positive Look Ahead. Това, което сме мачнали, трябва преди него си да има това, което е в след <=

(?<=\\.)

А тази - Positive Look Behind - Това , което сме мачнали трябва да има след него си , това което е в [].

(?[A-Z])

Най-простите регекси са литералите. Тоест търсим, нещо което е точно определено и искаме да го мачнем. Просто искаме да мачнем нещо 1 към 1. Нямаме патърни или някакви специални знаци, просто една думичка , която искаме да намерим в текста. В примера долу търсената думичка е регекс.

Exact Matching

- The simplest form of regex matching

regex

A **regular expression**, **regex** or **regexp** (sometimes called a **rational expression**) is, in **theoretical computer science** and **formal language theory**, a sequence of **characters** that define a **search pattern**.

Как работят регексите ?

Тръгваме от самото начало на текста и сравняваме първата буква на текста с първата буква на литерала. (Показан е на следващата снимка). В случая с картинката нямаме съвпадение. Отива на следващия знак (втория) и пак го сравняваме с първата буква на регекса. Ако отново няма съвпадение отива на третата буква от текста и така докато съвпадат всички знаци от търсения шаблон с дума от текста. Ако съвпадат едва тогава се мачва думата.

Exact Matching

- The simplest form of regex matching

regex

A **regular expression**, **regex** or **regexp** (sometimes called a **rational expression**) is, in **theoretical computer science** and **formal language theory**, a sequence of **characters** that define a **search pattern**.

КОГАТО ТЪРСИМ НЯКАКЪВ РЕГЕКС БЕЗ НЯКАКВИ СПЕЦИАЛНИ СИМВОЛИ (ПРИМЕРНО НЯМАМЕ СПЕЦИАЛНИЯ СИМВОЛ '.', КОЙТО МАЧВА ВСИЧКО), СЕ НАРИЧА ЛИТЕРАЛ. (НЕЩО, КОЕТО МАЧВАМЕ ЕДНО КЪМ ЕДНО). ТОВА Е НАЙ-ПРОСТИЯ ВИД РЕГЕКС.

Следващия вид регекс е така наречения шаблон за търсене (Pattern matching).

Използва се, когато търсим нещо по-сложно. Тоест не искаме да изреждаме всички думи, които търсим, а да намерим само

една дума.

Java Advanced - Regular Expressions - януари 2017 - Петър Пенев

Pattern Matching

SOFTWARE UNIVERSITY FOUNDATION

- Search patterns describe what should be matched

```
\+359[0-9]{9}
```

```
+61948228831222 - Dick
```

Какво представлява този шаблон ?

Имаме някои специални символи. Примерно в горната снимка - '[' и ']' , които ни казват, че това е някакъв клас. След това имаме т.нар. къдрави скоби - '{', '}' , които ни казват, че имаме някакъв оператор за повторение. И така.. този регекс започва с ескейпване(В регексите това става чрез символа '\') на знака '+', защото е специален символ. Когато е ескейпнат знака '+', той става литерал , който се търси едно към едно в текста, а не изпълнява функцията си да намира 1 или повече съвпадения както когато не е ескейпнат и е специален знак. След + , следват 3 цифри - 359 , които искаме да мацнем като литерали(1 към 1) , а не като шаблон. След тези цифри идва един клас - [0-9] , в който казваме , че искаме да мацнем една от тези цифри, които са вътре. В случая едно от числата от 0 до 9. Всичко , което е между тези [0-9], казва че искаме да намерим една от тези цифри. След класа идва операторът за повторение - {9}. Той казва, че искаме това, което седи преди оператора за повторение искаме да се повтаря точно 9 пъти.

В примера регекса работи по следния начин:

Тръгваме 1 по 1 да гледаме знаците. Започваме от първия знак. В случая и двата знака са + .След това продължаваме към трите цифри, които искаме да мацнем - 359 и веднага виждаме, че ги няма и затова маца въобще не продължава да сравнява нататък и спира докдето е стигнал и просто не намира нищо.

- Search patterns describe what should be matched

```
\+359[0-9]{9}
```

```
+61948228831222 - Dick
```

След това гледаме следващия пример:

```
+2394818322 - Matt
```

тук имаме едно +, но след това нищо не съвпада.

```
+3598418 2838 - Steven
```

тук + съвпада и цифрите съвпадат, но нататък ние сме казали , че искаме да имаме точно 9 цифри , а тук имаме 4 цифри и 1 space , който разваля patterna и нататък няма да сработи.

```
+3598418 2838 - Steven
```


В следващия пример вече всичко съвпада. Имаме +, 3те цифри - 359 и след тях точно 9 цифри.

+359882021853 – Andy

Има и още един интересен случай. Какво би станало ако имаме повече от 9 цифри както на следващата картинка.

+3598969233125321 – Nash

В този случай ще мачне точно толкова символа колкото му трябва според шаблона и останалото няма да го включи

Как се използват регексите ?

Най-базовият вариант е показан на следващата снимка.

```
Pattern pattern = Pattern.compile("a");
```

Имаме клас pattern, който ни компилира шаблона, който сме си направили. В случая литералът 'a'.

След това има още един клас -matcher.(Създава ни обект, който ще съдържа информация в себе си за всички мачове, които намери). Създава се, чрез обекта pattern, който сме си направили. Трябва да извикаме един статичен метод - matcher, и на него да подадем вече текста, в който искаме вече да търсим.

```
Matcher matcher = pattern.matcher("aaaab");
```

И сега ако решим, че искаме да намерим абсолютно всички съвпадения в текста "aaaaab", можем да извъртим един while цикъл докато намира нещо. В случая matcher.find() връща true, ако намери нещо и false, ако не намери, и по този начин цикълът while ще се върти докато е true, тоест докато има съвпадения.

```
Pattern pattern = Pattern.compile("a");  
Matcher matcher = pattern.matcher("aaaab");  
  
while (matcher.find()) {  
    System.out.println(matcher.group());  
}
```

Важно е да се знае, че нулевата група на един мач е целият мач. .group(); методът връща съдържанието на регекса. В конкретния пример резултатът ще бъде четири пъти 'a'.

Класове - Мачват само един , от многото символи които сме изредили вътре.

compact dis[ck]

Character Classes

Match One of Several Characters

Примерно имаме следния пример:

Java Advanced - Regular Expressions - януари 2017 - Петър Пенев

Character Classes

[aeiouy] – matches a lowercase vowel

Abraham Lincoln

Имаме следния клас [aeiouy], който съдържа всичките гласни в английския език. И сега искаме да потърсим в текста долу всяка една гласна от английската азбука. Този пример би върнал като резултат aaio. В примера A не се мачва, защото е главна буква, а ние сме задали в класа само малки букви. Ако искаме да хваща и големи букви, класът трябва да изглежда така - [AEIOUYaeiouy]. Така вече ще хваща и малки и големи букви.

[aeiouy] – matches a lowercase vowel

Abraham Lincoln

В класотите може да има обхват(range). Примерно следния клас [0123456789] може да се представи като [0-9], което е по-съкратено и по-лесно за четене. И с буквите е подобно - [A-Z] или [a-z].

Също така в класовете може да има множество range-ve - [A-Za-z] или [0-9a-z], [a-zA-z0-9]

Има и един специален символ - '.'

. Matches any symbol

Abraham Lincoln

Хваща всички символи с изключение на новите редове. Тях не ги хваща!

В класовете можем да слагаме и следния символ - '^'.

- # Abraham Lincoln

Abraham Lincoln

Java Advanced - Regular Expressions - януари 2017 - Петър Пенев

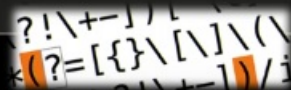
Shorthand Character Classes



- The is year 2033.

- The is year 2033.

- The is year 2033.



За тези съкращения също важат negation-те, само че не се правят със знака колибка, а с главни букви.

Negated Shorthand Character Classes

- `\D` – Shorthand for `[^0-9]`

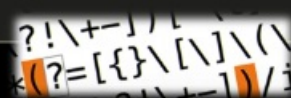
The is year 2033.

- `\W` – Shorthand for `[^a-zA-Z0-9_]`

The is year 2033.

- `\S` – Matches any non white-space character

The is year 2033.



Quantifiers - Repetition operators - Оператори за количество.

Quantifiers

- `+` - Matches the previous element one or more times

`\+[0-9]+`

`+359885976002`

`+`

Има няколко на вид такива оператора. Единият е '+'. С него казваме, че символа(в случая на горната снимка това е класът [0-9]. Класът се води за един единствен символ), който искаме да мачнем преди него искаме да се повтаря един или повече пъти. Затова в първия пример има мач, но не и във втория, защото там се мачва единствено знака +, класа за числа не хваща нищо.

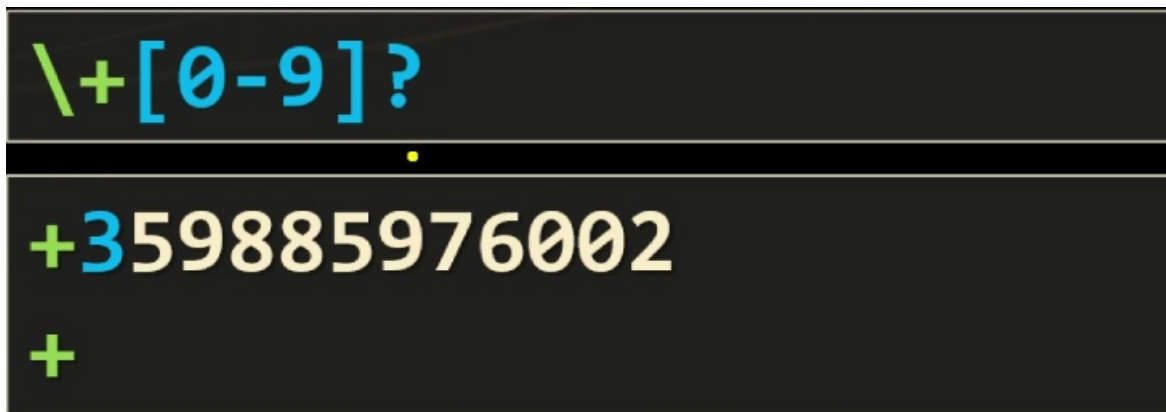
Друг оператор за количество е символът '*'. Той за разлика от '+' мачва 0 или повече повторения, тоест може да го има, но може и да го няма.

`\+[0-9]*`

`+359885976002`

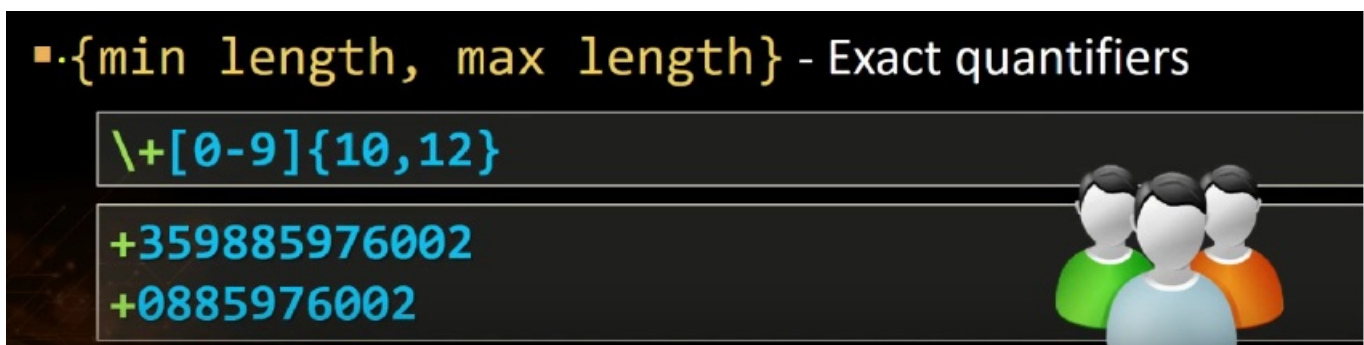
`+`

Друг оператор за количество е - '?'



Отнася се за предния символ, в случай за символ евентуално мачната от класа `[0-9]`. И казва, че искаме да имаме 0 или 1 повторения от това нещо. Не повече както е `'+'` и `'**'`, а точно 0 или 1. В примера на горната картинка хващаме `+` и след него точно една цифра. Във втория пример(където е само `+`) отново имаме мачване на `+` и след това нямаме цифри, но тъй като `'?'` казва 0 или 1, това е валиден мач.

И последният вид quantifiers са `{}`:



В `{}` можем да имаме минимална и максимална дължина. С този оператор казваме между колко и колко пъти искаме да се повтаря класа(конкретно в случая на горната снимка). В `\+[0-9]{10,12}` сме казали, че искаме една цифра да се повтаря между 10 и 12 пъти. В долните два примера цифрите са между 10 и 12 и затова се мачват.

В горния пример ако регекса се напише по следния начин `\+[0-9]{10, }` това ще мачне всичко от 10 цифри нагоре. Тоест вторият параметър може да се пропусне. Ако обаче се пропусне първия - `\+[0-9]{ , 10}`, това вече не е част от синтаксиса и ще търси в текста конкретно за `{ , 10}`. Ако искаме да търсим точно определена брой цифри може стане като оставим в `{}` само 1 параметър - `\+[0-9]{3}`, това би търсило за точно 3 цифри.

Когато се работи с quantifiers трябва да се има предвид нещо много важно. Quantifier-те мога да бъдат *lazy*(мързеливи) и *greedy*(алчни). По подразбиране са *greedy*.

- *greedy* - гледат да хванат колкото могат повече символи.

- Quantifiers are **greedy** by default

`"\."+`

Greedy repetition

Text `"with" some "quotations".`

- Make a quantifier **lazy** with ?

`"\.+?"`Text `"with" some "quotations".`

В горния пример - `"\."` , очакваме да ни хване два мача (`"width"` и `"quotations"`). Само че то тръгва и намира първите кавички и след това понеже сме му казали `.'` то ще продължава. На снимката в горния пример се вижда как вторите и третите кавички са в лилаво. Това е защото символът `'.'` ги хваща и понеже символът `'+'` е алчен ще хване и тях както е показано на следващата снимка.

`"with" some "quotations".`

И ще продължи нататък до края на регекса, след това като види че няма нищо ще се върне назад и ще види втората кавичка и ще разбере , че това е валиден мач.

`"\."`Text `"with" some "quotations".`

И ако имаме още думи в кавички, щеше да ги хване като един единствен мач.

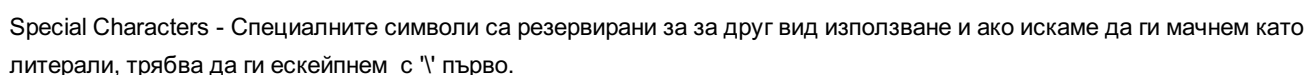
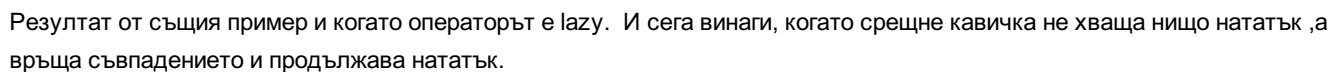
Как можем да управляваме тази особеност ?

- Като направим операторът от `greedy`(алчен) на `lazy`(мързелив)

- Make a quantifier **lazy** with ?

`"\.+?"`Text `"with" some "quotations".`

На следващата картинка е показан резултата , когато операторът е greedy.



Примерно символът '.' хваща абсолютно всичко , но ако искаме да намерим съпадение с '.' трябва този символ първо да се експейтне с '\\'.

Java Advanced - Regular Expressions - януари 2017 - Петър Пенев

regular expressions 101

@regex101

no match, 0

\. matches the character **.** literally
(case sensitive)

/ sentence \.

TEST STRING

SWITCH TO UI

sentence

Друг специален символ е '|' (pipe) - логическо или. Тоест когато искаме да мачнем едно или друго нещо.

■ | - Pipe is a logical OR

`\+359(|-) .+`

`+359 885/97-60-02`
`+359-885/97-60-02`
`+359/885/97-60-02`

Примерно искаме да мачнем някакъв телефонен номер, който започва с 359, след което има празно място или - (| -) това цялото е един знак и след него искаме всякакви знаци повторени 1 или повече пъти.

Друг специален символ са () - които обозначават група.

Anchors - котви.

Java Advanced - Regular Expressions - януари 2017 - Петър Пенев

Anchors


SOFTWARE UNIVERSITY FOUNDATION

■ ^ - The match must start at the beginning of the string or line

■ \$ - The match must occur at the end of the string or before `\n`

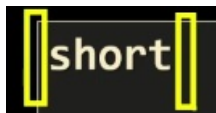
`^\w{6,12}$`

short

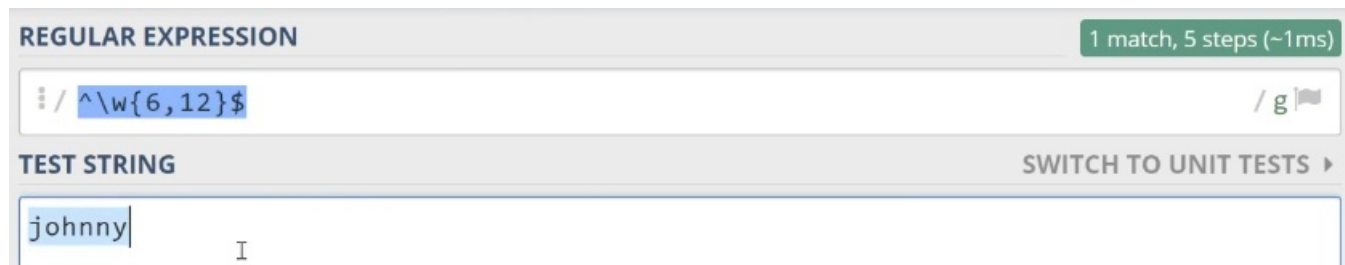


Двата знака - '^', '\$' указват началото и края на даден string. Тоест, за да може един match да е правилен трябва символът

'^' да е най-отпред и съответно стрингът трябва да завършва с '\$'. Не трябва да има нищо след това. Началото и краят на стринга хващат празното пространство преди думата, както е показано на следващата картинка. Празните пространства са символи с нулева дължина, но все пак присъстват там. В примера долу думата short не се мачва, защото `w{6, 12}` указва че трябва думата да има минимум 6 и максимум 12 знака. В този случай минимумът не покрива.



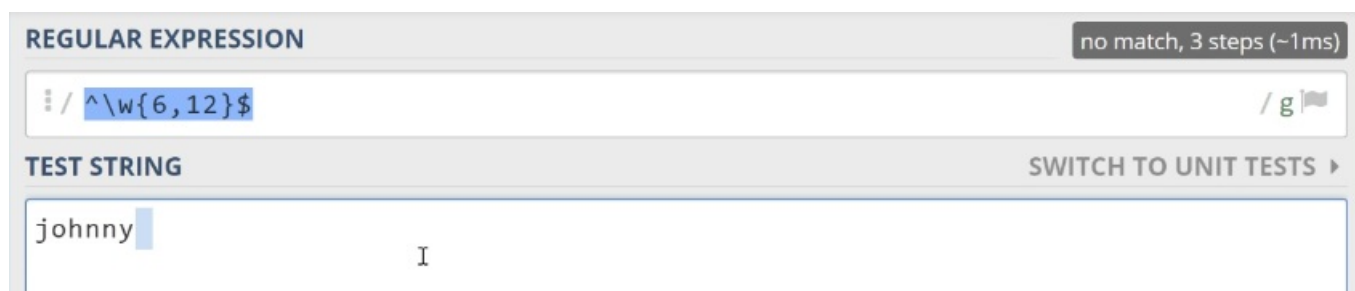
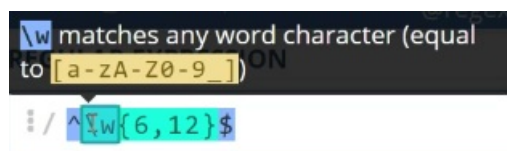
В следващия пример е показано как `pattern`-а мачва думата, написана по този начин:



REGULAR EXPRESSION: `^\\w{6,12}$` 1 match, 5 steps (~1ms)

TEST STRING: johnny

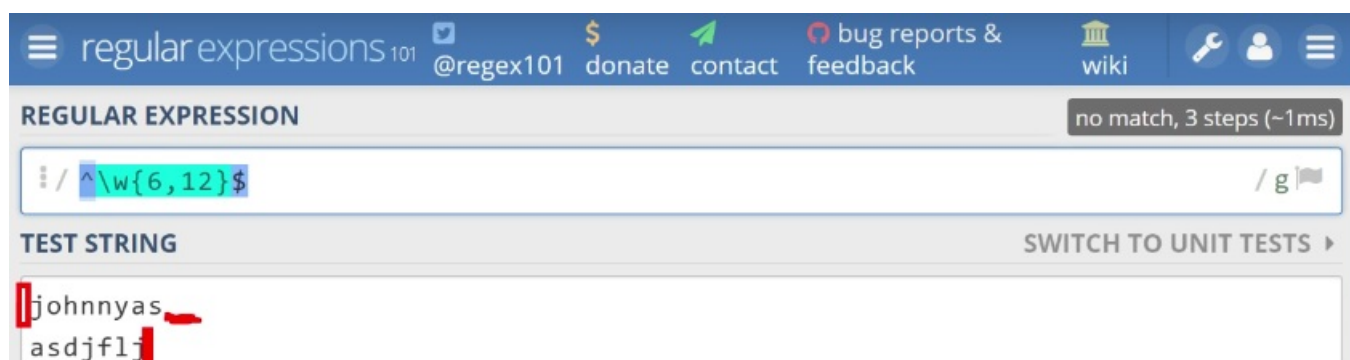
Само че ако след думата примерно добави празно място(space), тя няма да се мачне, защото `\\w` не включва в себе си символа space.



REGULAR EXPRESSION: `^\\w{6,12}$` no match, 3 steps (~1ms)

TEST STRING: johnny

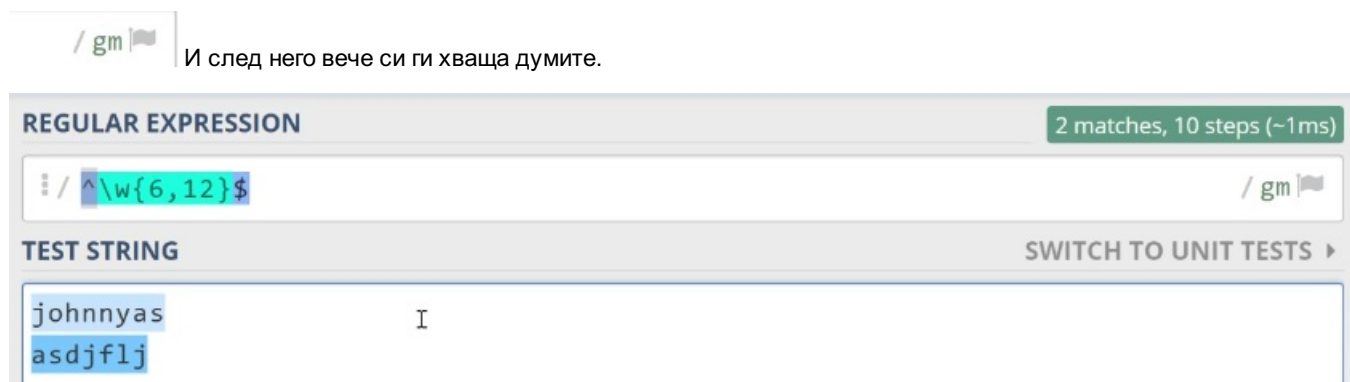
В следващия пример е показано, че `pattern`-а не мачва думите, защото '^' и '\$' не хващат символа за нов ред.



REGULAR EXPRESSION: `^\\w{6,12}$` no match, 3 steps (~1ms)

TEST STRING: johnnyas
asdjflj

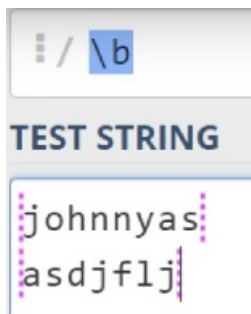
Ако искаме да направим така че да ни хваща всеки ред като отделен стринг трябва да добавим modifier `m`(multyline).



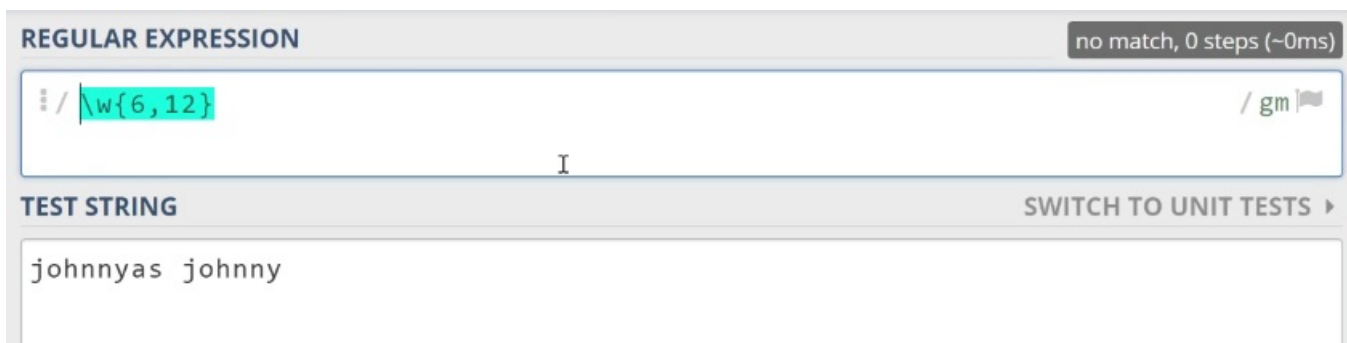
REGULAR EXPRESSION: `^\\w{6,12}$` 2 matches, 10 steps (~1ms)

TEST STRING: johnnyas
asdjflj

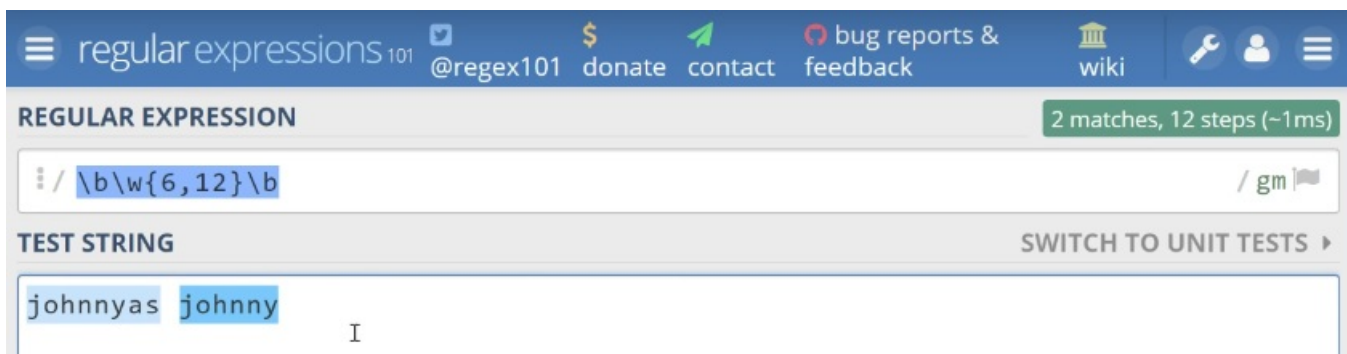
Друг символ, който е подобен е word bounty - '\b'. Това е граница на думата.



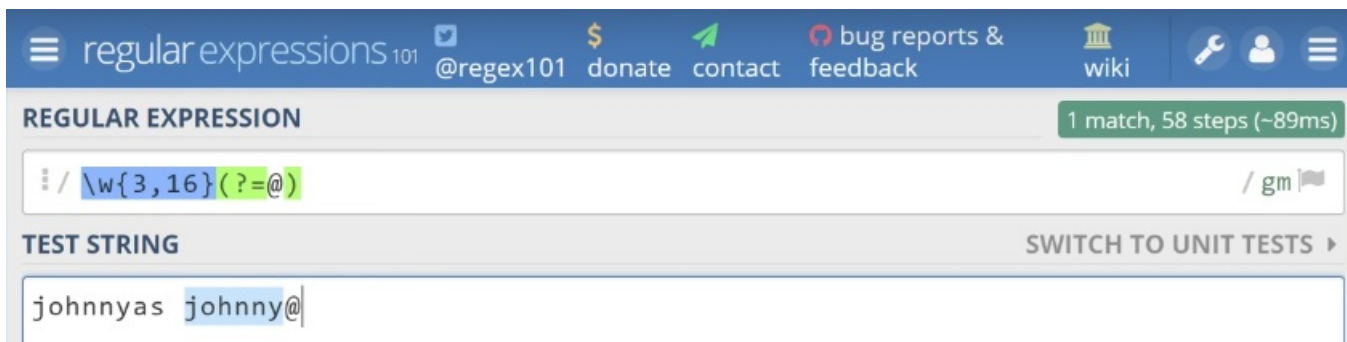
По същия начин като котвите, които хващаха началото и края на стринга, word bounty-to е по същия начин. Хваща граница на дума, която е с дължина 0. Използва се когато искаме да хванем примерно валиден username от някъде. Примерно знаем, че е част от стринга, а не че започва и свършва с него. И в този случай, когато е разположен измежду други думи трябва да го намерим по някакъв начин. И това можем да го направим с word bounty. В следващия пример искаме и двете думи, които са на един и същи ред да бъдат махнати.



И за да може да хване думите, patterna трябва да бъде както е показано на следващата картинка



В случаите когато искаме да махнем нещо преди или след въпросния мач се използват look ahead или look behind. Примерно искаме да хванем някакъв username с дължина между 3 и 16 символа, обаче искаме след него да няма други символи, освен тези които може да съдържа той. И за целта правим конструкция, която се нарича positive look ahead. Тоест искаме след този регекс да имаме нещо точно определено. В следващия пример искаме след регекса да имаме символа '@' и конструкцията ще хване само регекса, след който има '@'.



Ако искаме след регекса да няма '@' , трябва да сложим символа '!' между '?' и '=' . Това се нарича negative look ahead.

The screenshot shows a web interface for regular expressions. The top navigation bar includes links for 'regular expressions 101', '@regex101', 'donate', 'contact', 'bug reports & feedback', and 'wiki'. The main content area is titled 'REGULAR EXPRESSION' and shows the pattern `\w{3,16}(?!@)` with a tooltip 'Negative Lookahead'. Below the pattern, it indicates '2 matches, 11 steps (~1ms)'. The 'TEST STRING' section shows the text 'johnnyas johnny@'.

Следващия пример показва Look ahead накъде гледа.

The screenshot shows a web interface for regular expressions. The top navigation bar includes links for 'regular expressions 101', '@regex101', 'donate', 'contact', 'bug reports & feedback', and 'wiki'. The main content area is titled 'REGULAR EXPRESSION' and shows the pattern `\w{3,16}(?=@)`. Below the pattern, it indicates '2 matches, 11 steps (~1ms)'. The 'TEST STRING' section shows the text 'johnnyas johnny\$'. A red arrow points from the end of the second match to the end of the string, indicating the lookahead.

Докато Look behind -> (?<= \.)

The screenshot shows a web interface for regular expressions. The top navigation bar includes links for 'regular expressions 101', '@regex101', 'donate', 'contact', 'bug reports & feedback', and 'wiki'. The main content area is titled 'REGULAR EXPRESSION' and shows the pattern `\w{3,16}(?<= \.)`. Below the pattern, it indicates '2 matches, 11 steps (~1ms)'. The 'TEST STRING' section shows the text 'johnnyas johnny\$'. A red arrow points from the end of the second match back to the end of the string, indicating the lookbehind.

Конструкции - нещо, което не се прави с един символ , а с няколко. Такива конструкции са групите. Те представляват pattern, който е обграден от обикновени скоби - (). Тоест всичко, което сложим в такива скоби ние заявяваме , че искаме то да принадлежи към някаква група. Всичко, което е в такива скоби е отделна група. Примерно на снимката долу име 3 групи - една за деня, втора за месеца и трета за годината. От номера на самата група ние може да вземем съдържанието и.

The slide is titled 'Java Advanced - Regular Expressions - януари 2017 - Петър Пенев' and 'Grouping Constructs'. It features the 'SOFTWARE UNIVERSITY FOUNDATION' logo. The main content is a bullet point: '▪ (subexpression) - Captures a numbered group'. Below this, there are two examples of regular expressions: `(\d{2})-(\w{3})-(\d{4})` and `22-Jan-2015`.

▪ **(subexpression)** - Captures a numbered group

```
(\d{2})-(\w{3})-(\d{4})
```

```
22-Jan-2015
```

Group 0 = 22-Jan-2015

Group 1 = 22

Group 2 = Jan

Group 3 = 2015

Групите могат да се именуват. Ако не им зададем име те се нумерират с последователни числа.

▪ **(?<name>subexpression)** - Captures a named group

```
\d{2}-(?<month>\w{3})-\d{4}
```

```
22-Jan-2015
```