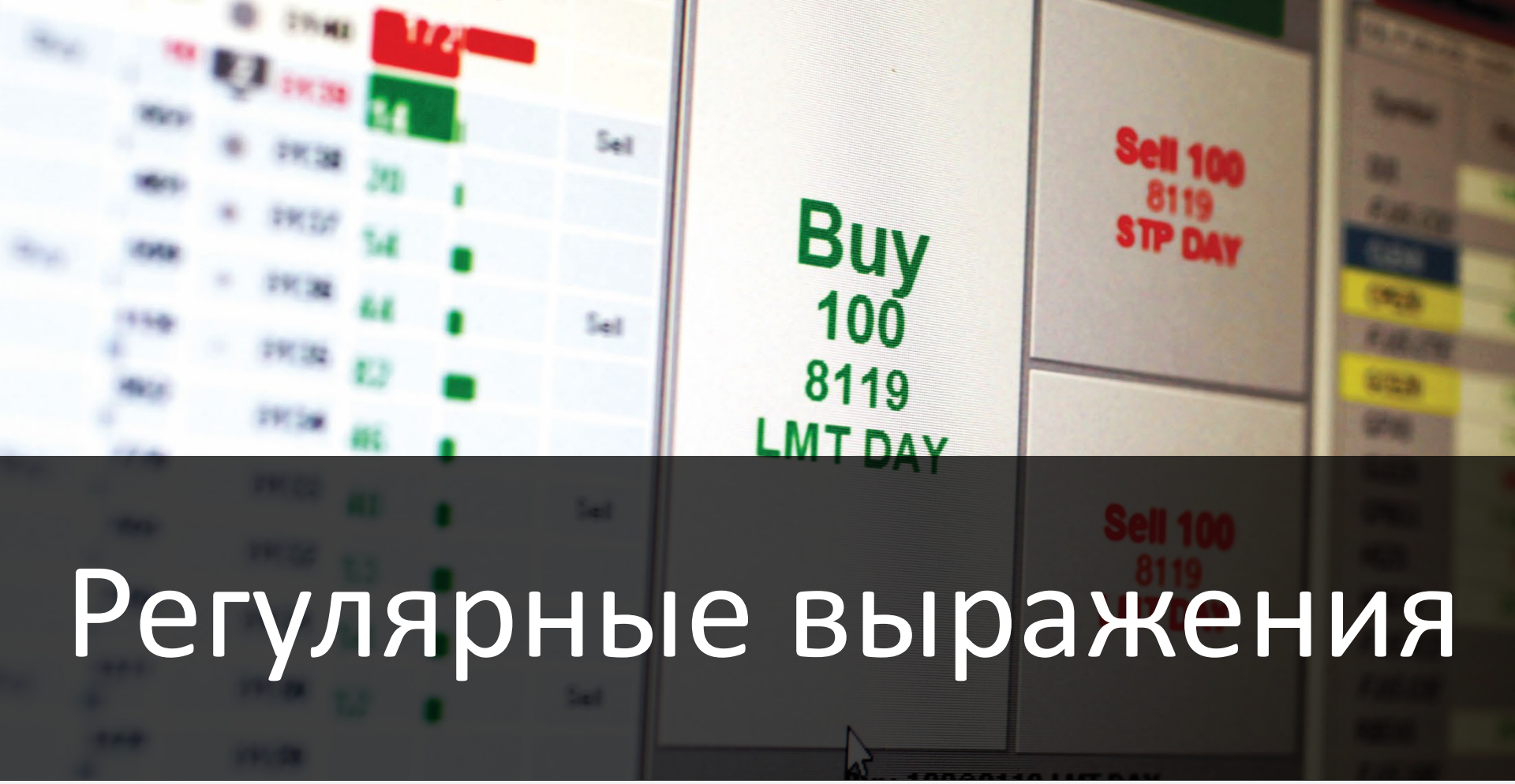


Регулярные выражения

Шипицын Кирилл Васильевич

15.12.2021

CQG

Возможности

- Проверка текста на соответствие шаблону
- Поиск текста по шаблону
- Извлечение текста по шаблону
- Замена текста по шаблону

Виды регулярных выражений

Виды синтаксиса:

- BRE = POSIX Basic Regular Expressions
- ERE = POSIX Extended Regular Expressions
- PCRE = Perl-Compatible Regular Expressions
- C++, .NET, JavaScript, Ruby, Java, везде разный синтаксис

Основы регулярных выражений

Алфавит – Unicode

Служебные символы – `[]{}()|\.+*?^$`

Экранирование	`\<sym>`
Конкатенация	`<first><second>`
Чередование	`<first>|<second>`
Группировка	`(<first>)`
Любой символ	`.`

Пример

```
#include (<|").....\.(h|hpp)("|>)
```

Пример

```
#include (<|").....\.(h|hpp)("|>)
```

Пример

```
#include (<|").....\.(h|hpp)("|>)
```

CQG

Пример

```
#include (<|").....\.(h|hpp)("|>)
```

Пример

```
#include (<|").....\.(h|hpp)("|>)
```

Пример

```
#include (<|").....\.(h|hpp)("|>)
```

Пример

```
#include (<|").....\.(h|hpp)("|>)
```

Invalid	Valid
#include<stdlib.h>	#include <stdlib.h>
#include "stdio.h"	#include "stdlib.h"
#include 'helper.hpp'	#include <helper.hpp>
#include "HELPER.HPP"	#include "helper.hpp"

Символьные классы

`[abcde]` – любой из перечисленных символов

`[^abcde]` – любой символ кроме перечисленных

`[a-t]` – задание диапазона символов

Класс	Эквивалент	Описание	Отрицание
\d	[0-9]	Цифры	\D
\s	[\f\n\r\t\v]	Пробелы	\S
\w	[a-zA-Z0-9_]	Буква, цифра, подчеркивание	\W

Пример

```
#include (<|")\w\w\w\w\w\w\.(h|hpp)("|>)
```

Пример

```
#include (<|")\w\w\w\w\w\w\.(h|hpp)("|>)
```

Пример

```
#include (<|")\w\w\w\w\w\w\.(h|hpp)("|>)
```

Invalid	Valid
#include <std.io.h>	#include <stdlib.h>
#include "std-io.h"	#include "stdlib.h"
#include <ab/csv.hpp>	#include <helper.hpp>
#include "../csv.hpp"	#include "helper.hpp"

CQG

Квантификаторы

Синтаксис: `<expr><quant>`

Ищется **максимальное** вхождение

`{n}`	Ровно n раз
`{m,n}`	От m до n раз
`{m,}`	Не менее m раз
`{,n}`	Не более n раз
`{0,1}` = `?`	0 или 1 раз
`{0,}` = `*`	0 и более раз (любое количество раз)
`{1,}` = `+`	1 и более раз

Пример

```
#include[ \t]*
(<|")[\.\\\/:\w]+\.(h|hpp)("|>)
```

Пример

```
#include[ \t]*
(<|")[\.\\\/:\w]+\.(h|hpp)("|>)
```

Пример

```
#include[ \t]*
(<|")[\.\\\/:\w]+\.(h|hpp)("|>)
```

Пример

```
#include[ \t]*
(<|")[\.\\\/:\w]+\.(h|hpp)("|>)
```

Valid
#include "C:\\Users\\Admin\\Desktop\\project\\helper.hpp"
#include "../helper.hpp"
#include "project/helper.hpp"
#include<stdio.h>

CQG

Символы позиционирования

Позиционируют регулярное выражение в строке относительно начала/конца строки/слова.

^	Начало текста	^cat	catdog catdog
$	Конец текста	dog$	catdog catdog
\b	Граница слова	\bcat	catcat catcat
\B	Не граница слова	\Bcat	catcat catcat

Модификаторы

`(?<mod>)`, `(?-<mod>)` – в начале выражения включает/выключает модификатор

Модификатор g(global)

Искать все вхождения

```
\d
```

g	-g
1 2 3 4 5	1 2 3 4 5

Модификатор m(multi-line)

^$ соответствуют началу/концу строк

```
^is$
```

m	-m
is this his	is this his

Модификатор `i(insensitive)`

Отключить чувствительность к регистру

```
[a-z]
```

`i`	`-i`
a b C D E	a b C D E

Модификатор x(extended)

Можно писать комментарии после #. Пробелы игнорируются.

```
\d #Цифры
\s #Пробелы
\w #Буква, цифра, подчеркивание
```

x	-x
1 b 2 c 3 d	1 b 2 c 3 d

CQG

Модификатор `s(single-line)`

Точка (`.`) соответствует символу новой строки

`s`	`-s`
123 456	123 456

Проблема

```
<td>.*<\/td>
```

```
<table>
<tr><th>Firstname</th><th>Lastname</th><th>Age</th></tr>
<tr><td>Jill</td><td>Smith</td><td>50</td></tr>
<tr><td>Eve</td><td>Jackson</td><td>94</td></tr>
</table>
```

CQG

Проблема

```
<td>.*<\/td>
```

```
<table>
<tr><th>Firstname</th><th>Lastname</th><th>Age</th></tr>
<tr><td>Jill</td><td>Smith</td><td>50</td></tr>
<tr><td>Eve</td><td>Jackson</td><td>94</td></tr>
</table>
```

CQG

Ленивые квантификаторы

Ищут минимальное вхождение.

Синтаксис:

```
<expr><quant>?
```

Решение

```
<td>.*?<\/td>
```

```
<table>
<tr><th>Firstname</th><th>Lastname</th><th>Age</th></tr>
<tr><td>Jill</td><td>Smith</td><td>50</td></tr>
<tr><td>Eve</td><td>Jackson</td><td>94</td></tr>
</table>
```

CQG

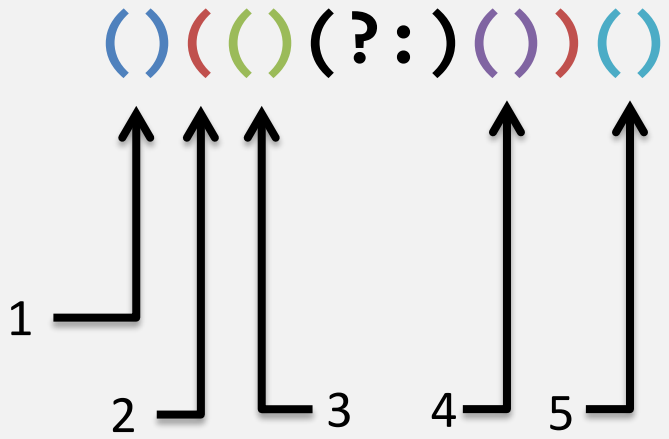
Группировка и захваты

`(<expr>)` – захватывающая группа

`(?:<expr>)` – незахватывающая группа

`(?<name><expr>)` – именованная группа

Нумерация групп (кроме именованных)



Пример

```
^(?:\+?[78][\- ]?)?
(\(?\d{3}\)?[\- ]?)?
([\d\- ]{7,10})$
```

Пример

```
^(?:\+?[78][\- ]?)?
(\(?\d{3}\)?[\- ]?)?
([\d\- ]{7,10})$
```

Пример

```
^(?:\+?[78][\- ]?)?
(\(?\d{3}\)?[\- ]?)?
([\d\- ]{7,10})$
```

Пример

```
^(?:\+?[78][\- ]?)?
(\(?\d{3}\)?[\- ]?)?
([\d\- ]{7,10})$
```

Пример

```
^(?:\+?[78][\- ]?)?
(\(?\d{3}\)?[\- ]?)?
([\d\- ]{7,10})$
```

Full match	+79261234567
Group 1	926
Group 2	1234567

Ссылки назад

В регулярном выражении можно ссылаться на уже совпавшие группы.

Синтаксис:

`\<n>, n∈[0;99]`	`\1, \2, …`
`\g{<n>}`	`\g{1}, \g{2}, …`
`\g{<name>}`	`\g{<a>}, \g{<b>}, …`

Пример

```
('|").*?\g{1}
```

Пример

```
('|").*?\g{1}
```

Пример

```
('|").*?\g{1}
```

Пример

```
('|").*?\g{1}
```

Пример

```
('|").*?\g{1}
```

Invalid	Valid
"abc'	"abc"
	""
	'abc'

Условный оператор

Проверяет было ли найдено соответствие для предыдущей группы по имени или номеру (либо для выражения нулевой длины).

Синтаксис:

```
(?(<name>)<first>|<second>)
(?(?=<expr>)<first>|<second>)
```

Пример

(?(<name>)<first>|<second>)

```
(?<comment>^\/\/)?
.*
(?(<comment>)(\n)|(;\n)$)
```

Пример (?(<name>)<first>|<second>)

```
(?<comment>^\/\/)?
.*
(?(<comment>)(\n)|(;\n)$)
```

Пример (?(<name>)<first>|<second>)

```
(?<comment>^\/\/)?
.*
(?(<comment>)(\n)|(;\n)$)
```

Пример (?(<name>)<first>|<second>)

```
(?<comment>^\/\/)?
.*
(?(<comment>)(\n)|(;\n)$)
```

Пример (?(<name>)<first>|<second>)

```
(?<comment>^\/\/)?
.*
(?(<comment>)(\n)|(;\n)$)
```

Пример (?(<name>)<first>|<second>)

```
(?<comment>^\/\/)?
.*
(?(<comment>)(\n)|(;\n)$)
```

Пример (?(<name>)<first>|<second>)

```
(?<comment>^\/\/)?
.*
(?(<comment>)(\n)|(;\n)$)
```

Invalid	Valid
int x = 0	int x = 0;
	// int x = 0
	// int x = 0;

CQG

Пример

(?(?=<expr>)<first>|<second>)

(?(?=^\/\/)(.*\n)|(.*;\n)$)

Пример

(?(?=<expr>)<first>|<second>)

(?(?=^\/\/)(.*\n)|(.*;\n)$)

Пример

(?(?=<expr>)<first>|<second>)

(?(?=^\/\/)(.*\n)|(.*;\n)$)

Пример

(?(?=<expr>)<first>|<second>)

(?(?=^\/\/)(.*\n)|(.*;\n)$)

Пример

(?(?=<expr>)<first>|<second>)

```
(?(?=^\/\/)(.*\n)|(.*;\n)$)
```

Invalid	Valid
int x = 0	int x = 0;
	// int x = 0
	// int x = 0;

CQG

Заглядывание вперед (выражение нулевой длины)

Просматривание последующего текста без включения в найденное.

Синтаксис:

`(?=<expr>)` – позитивное

`(?!<expr>)` – негативное

Пример (?=<expr>)

```
\d+(?=,)
```

```
5, 10, 15, 20, 25, 30
```

Пример (?=<expr>)

```
\d+(?=,)
```

```
5, 10, 15, 20, 25, 30
```

Пример (?!<expr>)

```
\d+(?!,)
```

```
5, 10, 15, 20, 25, 30
```

Пример (?!<expr>)

```
\d+(?!,)
```

```
5, 10, 15, 20, 25, 30
```

Оглядывание назад (выражение нулевой длины)

Просматривание предыдущего текста без включения в найденное.

Квантификаторы запрещены.

Синтаксис:

`(?<=<expr>)` – позитивное

`(?<!<expr>)` – негативное

Пример (?<=<expr>)

```
(?<=,\s)(\d+)
```

```
5, 10, 15, 20, 25, 30
```

Пример (?<=<expr>)

```
(?<=,\s)(\d+)
```

```
5, 10, 15, 20, 25, 30
```

Пример (?<!<expr>)

```
(?<!,\s)(\d+)
```

```
5, 10, 15, 20, 25, 30
```

Пример (?<!<expr>)

```
(?<!,\s)(\d+)
```

```
5, 10, 15, 20, 25, 30
```

Замена текста

Большинство программ позволяют производить замену по регулярным выражениям.

В замещающем тексте можно использовать захваченные группы.

Пример

Поиск	`(\d+)u`
Замена	`\1 или $1`

Исходный текст:	**Результат:**
const auto x = 1u;	const auto x = 1;

CQG

Пример std::regex_match

```
#include <iostream>
#include <string>
#include <regex>

int main()
{
    std::string fnames[] = {"foo.txt", "bar.txt", "zoidberg"};

    std::regex txt_regex("[a-z]+\\.txt");
    for (const auto &fname : fnames) {
        std::cout << fname << ": " << std::regex_match(fname, txt_regex) << '\n';
    }
}
```

ru.cppreference.com/w/cpp/regex/regex_match

Пример std::regex_match

```
#include <iostream>
#include <string>
#include <regex>

int main()
{
    std::string fnames[] = {"foo.txt", "bar.txt", "zoidberg"};

    std::regex txt_regex("[a-z]+\\.txt");
    for (const auto &fname : fnames) {
        std::cout << fname << ": " << std::regex_match(fname, txt_regex) << '\n';
    }
}
```

ru.cppreference.com/w/cpp/regex/regex_match

Пример std::regex_match

```
#include <iostream>
#include <string>
#include <regex>

int main()
{
    std::string fnames[] = {"foo.txt", "bar.txt", "zoidberg"};

    std::regex txt_regex("[a-z]+\\.txt");
    for (const auto &fname : fnames) {
        std::cout << fname << ": " << std::regex_match(fname, txt_regex) << '\n';
    }
}
```

ru.cppreference.com/w/cpp/regex/regex_match

Пример std::regex_match

```
#include <iostream>
#include <string>
#include <regex>

int main()
{
    std::string fnames[] = {"foo.txt", "bar.txt", "zoidberg"};

    std::regex txt_regex("[a-z]+\\.txt");
    for (const auto &fname : fnames) {
        std::cout << fname << ": " << std::regex_match(fname, txt_regex) << '\n';
    }
}
```

ru.cppreference.com/w/cpp/regex/regex_match

Пример std::regex_match

```
#include <iostream>
#include <string>
#include <regex>

int main()
{
    std::string fnames[] = {"foo.txt", "bar.txt", "zoidberg"};

    std::regex txt_regex("[a-z]+\\.txt");
    for (const auto &fname : fnames) {
        std::cout << fname << ": " << std::regex_match(fname, txt_regex) << '\n';
    }
}
```

```
foo.txt: 1
bar.txt: 1
zoidberg: 0
```

ru.cppreference.com/w/cpp/regex/regex_match

Пример std::regex_search

```
#include <iostream>
#include <string>
#include <regex>

int main()
{
   std::string s("this subject has a submarine as a subsequence");
   std::smatch m;
   std::regex e("\\b(sub)([^ ]*)");

   std::cout << "Target sequence: " << s << std::endl;
   std::cout << "Regular expression: /\\b(sub)([^ ]*)/" << std::endl;
   std::cout << "The following matches and submatches were found:" << std::endl;

   while (std::regex_search (s,m,e))
   {
      for (auto x:m) std::cout << x << " ";
      std::cout << std::endl;
      s = m.suffix().str();
   }

   return 0;
}
```

tutorialspoint.com/cpp_standard_library/cpp_regex_search.htm

CQG

Пример std::regex_search

```
#include <iostream>
#include <string>
#include <regex>

int main()
{
   std::string s("this subject has a submarine as a subsequence");
   std::smatch m;
   std::regex e("\\b(sub)([^ ]*)");

   std::cout << "Target sequence: " << s << std::endl;
   std::cout << "Regular expression: /\\b(sub)([^ ]*)/" << std::endl;
   std::cout << "The following matches and submatches were found:" << std::endl;

   while (std::regex_search (s,m,e))
   {
      for (auto x:m) std::cout << x << " ";
      std::cout << std::endl;
      s = m.suffix().str();
   }

   return 0;
}
```

tutorialspoint.com/cpp_standard_library/cpp_regex_search.htm

CQG

Пример std::regex_search

```
#include <iostream>
#include <string>
#include <regex>

int main()
{
   std::string s("this subject has a submarine as a subsequence");
   std::smatch m;
   std::regex e("\\b(sub)([^ ]*)");

   std::cout << "Target sequence: " << s << std::endl;
   std::cout << "Regular expression: /\\b(sub)([^ ]*)/" << std::endl;
   std::cout << "The following matches and submatches were found:" << std::endl;

   while (std::regex_search (s,m,e))
   {
      for (auto x:m) std::cout << x << " ";
      std::cout << std::endl;
      s = m.suffix().str();
   }

   return 0;
}
```

tutorialspoint.com/cpp_standard_library/cpp_regex_search.htm

CQG

Пример std::regex_search

```
#include <iostream>
#include <string>
#include <regex>

int main()
{
   std::string s("this subject has a submarine as a subsequence");
   std::smatch m;
   std::regex e("\\b(sub)([^ ]*)");

   std::cout << "Target sequence: " << s << std::endl;
   std::cout << "Regular expression: /\\b(sub)([^ ]*)/" << std::endl;
   std::cout << "The following matches and submatches were found:" << std::endl;

   while (std::regex_search (s,m,e))
   {
      for (auto x:m) std::cout << x << " ";
      std::cout << std::endl;
      s = m.suffix().str();
   }

   return 0;
}
```

tutorialspoint.com/cpp_standard_library/cpp_regex_search.htm

CQG

Пример std::regex_search

```
#include <iostream>
#include <string>
#include <regex>

int main()
{
   std::string s("this subject has a submarine as a subsequence");
   std::smatch m;
   std::regex e("\\b(sub)([^ ]*)");

   std::cout << "Target sequence: " << s << std::endl;
   std::cout << "Regular expression: /\\b(sub)([^ ]*)/" << std::endl;
   std::cout << "The following matches and submatches were found:" << std::endl;

   while (std::regex_search (s,m,e))
   {
      for (auto x:m) std::cout << x << " ";
      std::cout << std::endl;
      s = m.suffix().str();
   }

   return 0;
}
```

tutorialspoint.com/cpp_standard_library/cpp_regex_search.htm

CQG

Пример std::regex_search

```
#include <iostream>
#include <string>
#include <regex>

int main()
{
   std::string s("this subject has a submarine as a subsequence");
   std::smatch m;
   std::regex e("\\b(sub)([^ ]*)");

   std::cout << "Target sequence: " << s << std::endl;
   std::cout << "Regular expression: /\\b(sub)([^ ]*)/" << std::endl;
   std::cout << "The following matches and submatches were found:" << std::endl;

   while (std::regex_search (s,m,e))
   {
      for (auto x:m) std::cout << x << " ";
      std::cout << std::endl;
      s = m.suffix().str();
   }

   return 0;
}
```

tutorialspoint.com/cpp_standard_library/cpp_regex_search.htm

CQG

Пример std::regex_search

```
#include <iostream>
#include <string>
#include <regex>

int main()
{
   std::string s("this subject has a submarine as a subsequence");
   std::smatch m;
   std::regex e("\\b(sub)([^ ]*)");

   std::cout << "Target sequence: " << s << std::endl;
   std::cout << "Regular expression: /\\b(sub)([^ ]*)/" << std::endl;
   std::cout << "The following matches and submatches were found:" << std::endl;

   while (std::regex_search (s,m,e))
   {
      for (auto x:m) std::cout << x << " ";
      std::cout << std::endl;
      s = m.suffix().str();
   }

   return 0;
}
```

tutorialspoint.com/cpp_standard_library/cpp_regex_search.htm

CQG

Пример std::regex_search

```
Target sequence: this subject has a submarine as a subsequence
Regular expression: /\b(sub)([^ ]*)/
The following matches and submatches were found:
subject sub ject
submarine sub marine
subsequence sub sequence
```

tutorialspoint.com/cpp_standard_library/cpp_regex_search.htm

Пример std::regex_match

```
#include <iostream>
#include <regex>
#include <string>
int main()
{
   std::string text = "Quick brown fox";
   std::regex vowel_re("a|o|e|u|i");
   std::cout << std::regex_replace(text, vowel_re, "[$&]") << '\n';
}
```

ru.cppreference.com/w/cpp/regex/regex_replace

Пример std::regex_match

```
#include <iostream>
#include <regex>
#include <string>
int main()
{
   std::string text = "Quick brown fox";
   std::regex vowel_re("a|o|e|u|i");
   std::cout << std::regex_replace(text, vowel_re, "[$&]") << '\n';
}
```

ru.cppreference.com/w/cpp/regex/regex_replace

Пример std::regex_match

```
#include <iostream>
#include <regex>
#include <string>
int main()
{
   std::string text = "Quick brown fox";
   std::regex vowel_re("a|o|e|u|i");
   std::cout << std::regex_replace(text, vowel_re, "[$&]") << '\n';
}
```

ru.cppreference.com/w/cpp/regex/regex_replace

Пример std::regex_match

```
#include <iostream>
#include <regex>
#include <string>
int main()
{
   std::string text = "Quick brown fox";
   std::regex vowel_re("a|o|e|u|i");
   std::cout << std::regex_replace(text, vowel_re, "[$&]") << '\n';
}
```

ru.cppreference.com/w/cpp/regex/regex_replace

Пример std::regex_replace

```
#include <iostream>
#include <regex>
#include <string>
int main()
{
   std::string text = "Quick brown fox";
   std::regex vowel_re("a|o|e|u|i");
   std::cout << std::regex_replace(text, vowel_re, "[$&]") << '\n’;
}
```

```
Q[u][i]ck br[o]wn f[o]x
```

ru.cppreference.com/w/cpp/regex/regex_replace

Ссылки

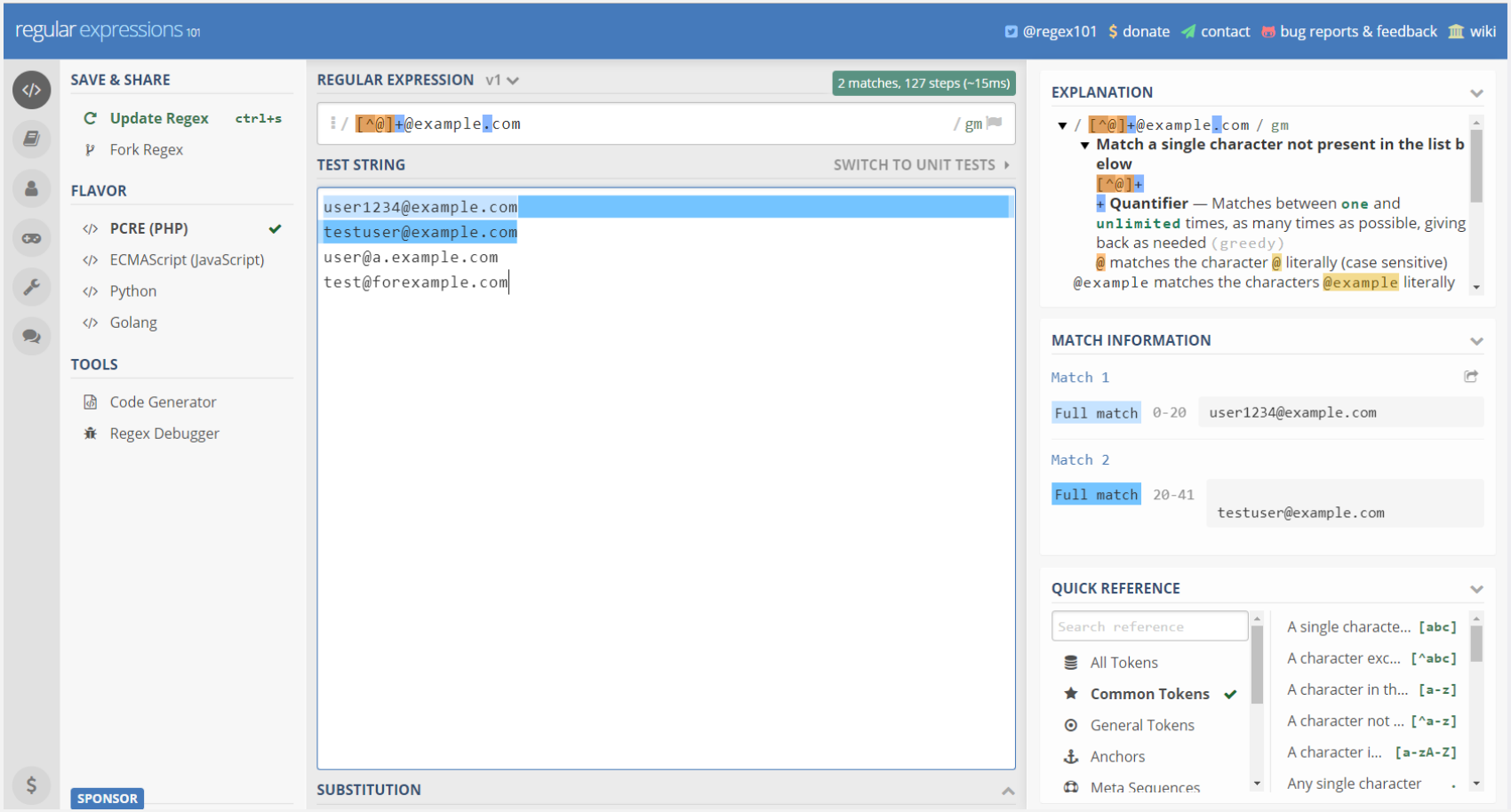
regex101.com

rexegg.com

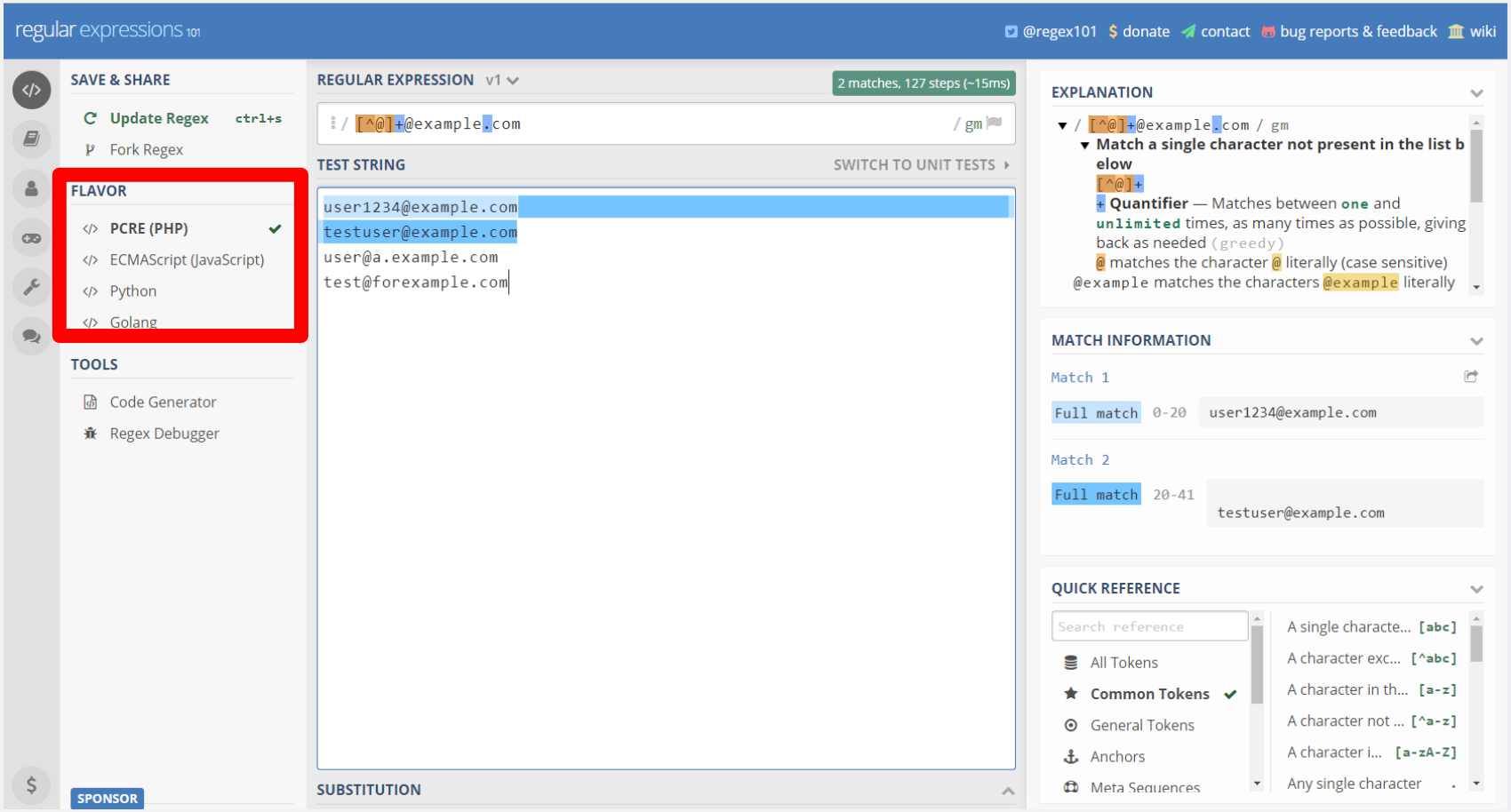
regular-expressions.info

habr.com/ru/post/166777/

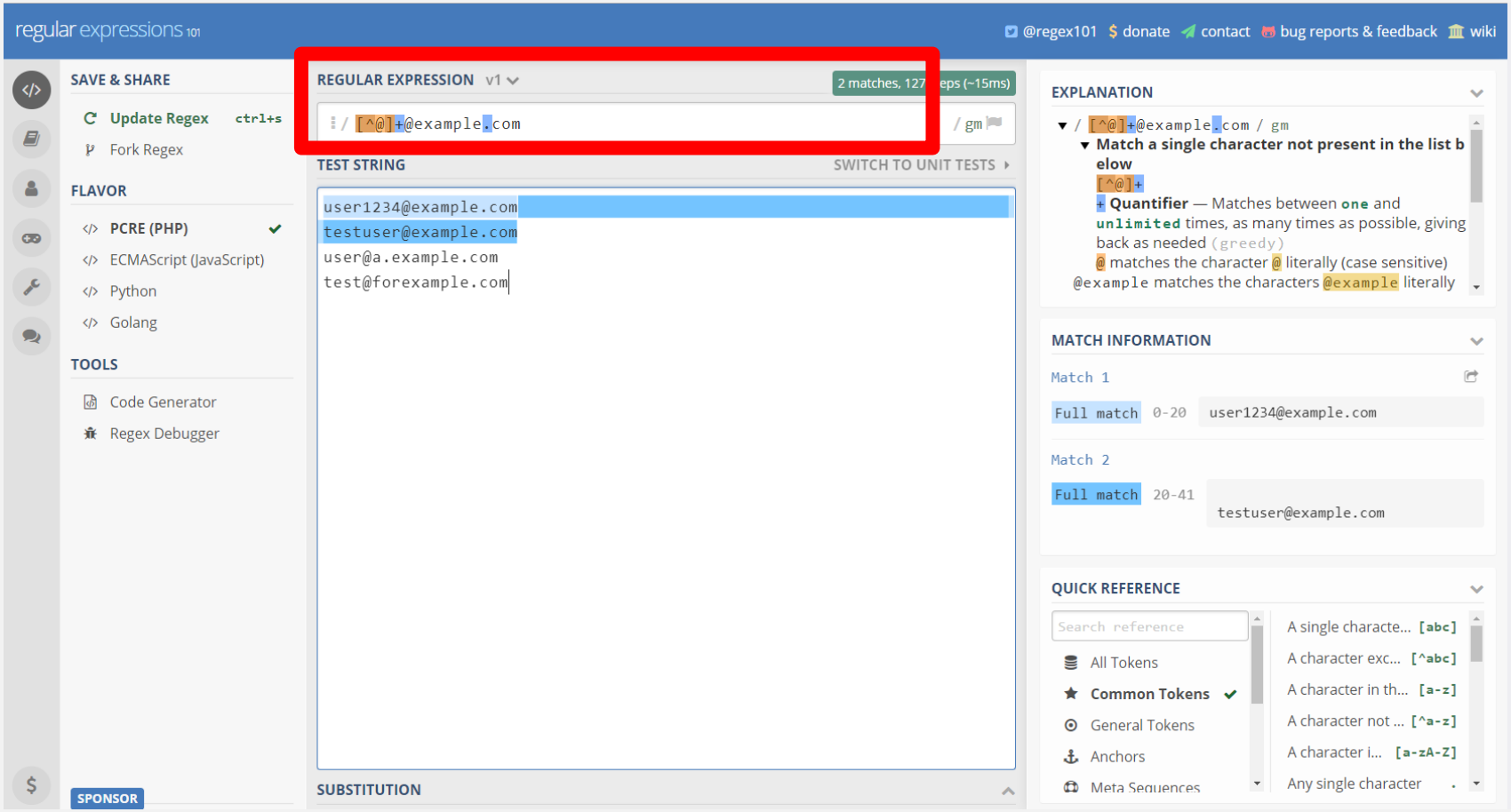
Regex101


regular expressions 101
@regex101
$ donate
contact
bug reports & feedback
wiki
SAVE & SHARE
Update Regex ctrl+s
Fork Regex
FLAVOR
PCRE (PHP)
ECMAScript (JavaScript)
Python
Golang
TOOLS
Code Generator
Regex Debugger
SPONSOR
REGULAR EXPRESSION v1
2 matches, 127 steps (~15ms)
/ [^@]+@example.com / gm
TEST STRING
SWITCH TO UNIT TESTS
user1234@example.com
testuser@example.com
user@a.example.com
test@forexample.com
SUBSTITUTION
EXPLANATION
/ [^@]+@example.com / gm
Match a single character not present in the list below
[^@]+
+ Quantifier — Matches between one and unlimited times, as many times as possible, giving back as needed (greedy)
@ matches the character @ literally (case sensitive)
@example matches the characters @example literally
MATCH INFORMATION
Match 1
Full match 0-20 user1234@example.com
Match 2
Full match 20-41 testuser@example.com
QUICK REFERENCE
Search reference
All Tokens
Common Tokens
General Tokens
Anchors
Meta Sequences
A single characte... [abc]
A character exc... [^abc]
A character in th... [a-z]
A character not ... [^a-z]
A character i... [a-zA-Z]
Any single character .

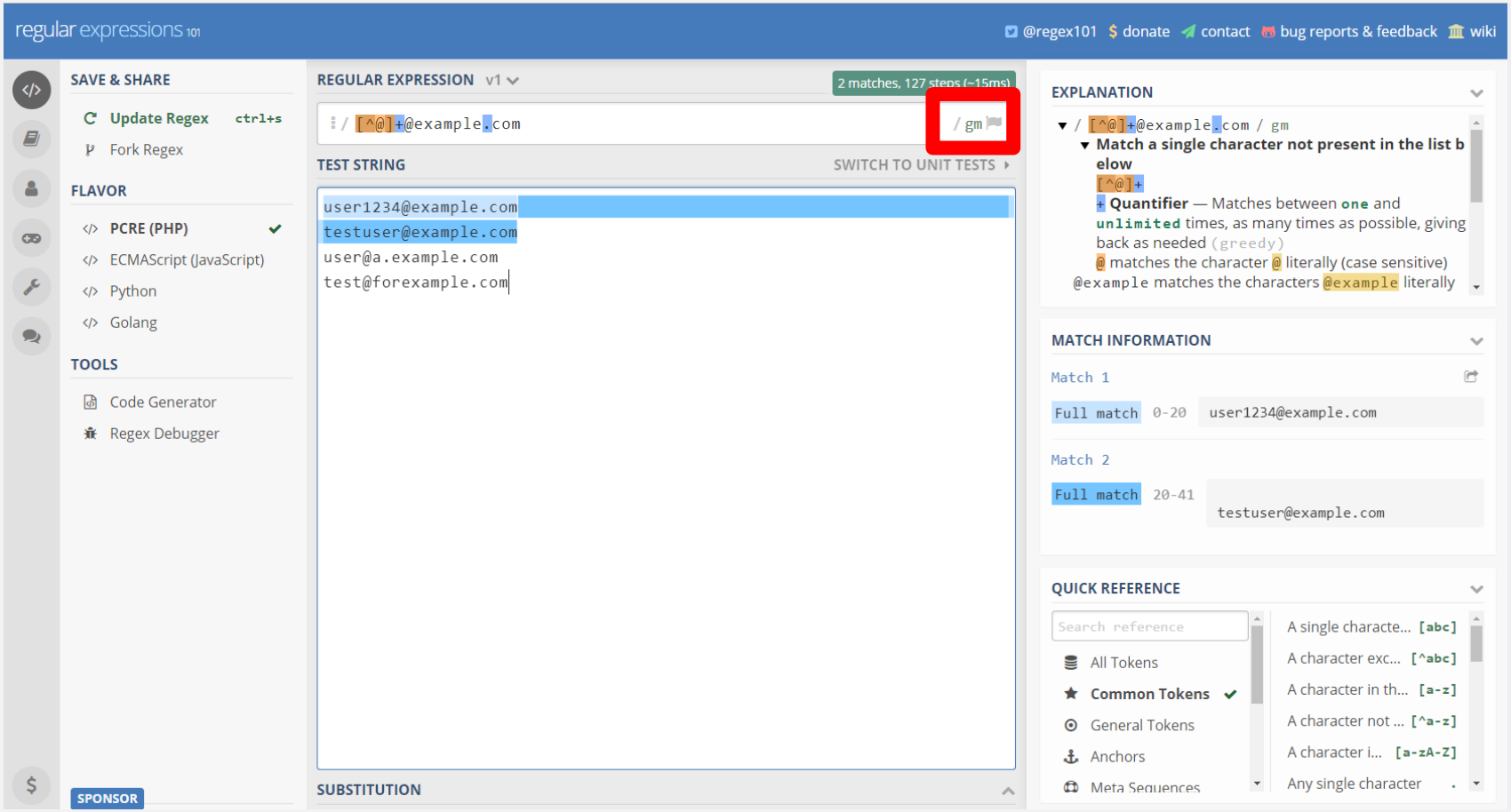
Regex101


regular expressions 101
@regex101
$ donate
contact
bug reports & feedback
wiki
SAVE & SHARE
Update Regex ctrl+s
Fork Regex
FLAVOR
PCRE (PHP)
ECMAScript (JavaScript)
Python
Golang
TOOLS
Code Generator
Regex Debugger
SPONSOR
REGULAR EXPRESSION v1
2 matches, 127 steps (~15ms)
/ [^@]+@example.com / gm
TEST STRING
SWITCH TO UNIT TESTS
user1234@example.com
testuser@example.com
user@a.example.com
test@forexample.com
SUBSTITUTION
EXPLANATION
/ [^@]+@example.com / gm
Match a single character not present in the list below
[^@]+
+ Quantifier — Matches between one and unlimited times, as many times as possible, giving back as needed (greedy)
@ matches the character @ literally (case sensitive)
@example matches the characters @example literally
MATCH INFORMATION
Match 1
Full match 0-20 user1234@example.com
Match 2
Full match 20-41 testuser@example.com
QUICK REFERENCE
Search reference
All Tokens
Common Tokens
General Tokens
Anchors
Meta Sequences
A single characte... [abc]
A character exc... [^abc]
A character in th... [a-z]
A character not ... [^a-z]
A character i... [a-zA-Z]
Any single character .

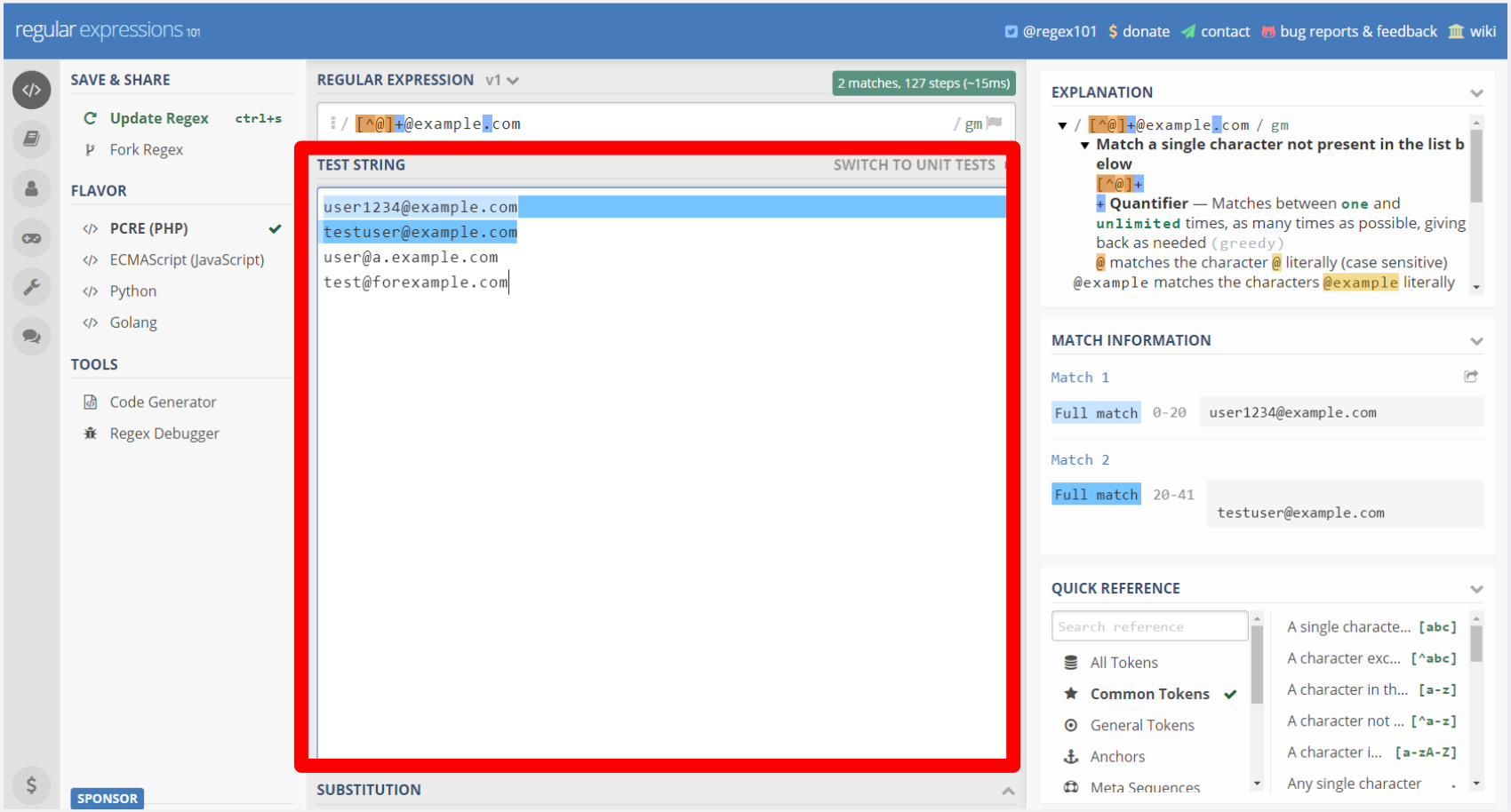
Regex101


regular expressions 101
@regex101
$ donate
contact
bug reports & feedback
wiki
SAVE & SHARE
Update Regex ctrl+s
Fork Regex
FLAVOR
PCRE (PHP)
ECMAScript (JavaScript)
Python
Golang
TOOLS
Code Generator
Regex Debugger
SPONSOR
REGULAR EXPRESSION v1
2 matches, 127 eps (~15ms)
/ [^@]+@example.com / gm
TEST STRING
SWITCH TO UNIT TESTS
user1234@example.com
testuser@example.com
user@a.example.com
test@forexample.com
SUBSTITUTION
EXPLANATION
/ [^@]+@example.com / gm
Match a single character not present in the list below
[^@]+
+ Quantifier — Matches between one and unlimited times, as many times as possible, giving back as needed (greedy)
@ matches the character @ literally (case sensitive)
@example matches the characters @example literally
MATCH INFORMATION
Match 1
Full match 0-20 user1234@example.com
Match 2
Full match 20-41 testuser@example.com
QUICK REFERENCE
Search reference
All Tokens
Common Tokens
General Tokens
Anchors
Meta Sequences
A single characte... [abc]
A character exc... [^abc]
A character in th... [a-z]
A character not ... [^a-z]
A character i... [a-zA-Z]
Any single character .

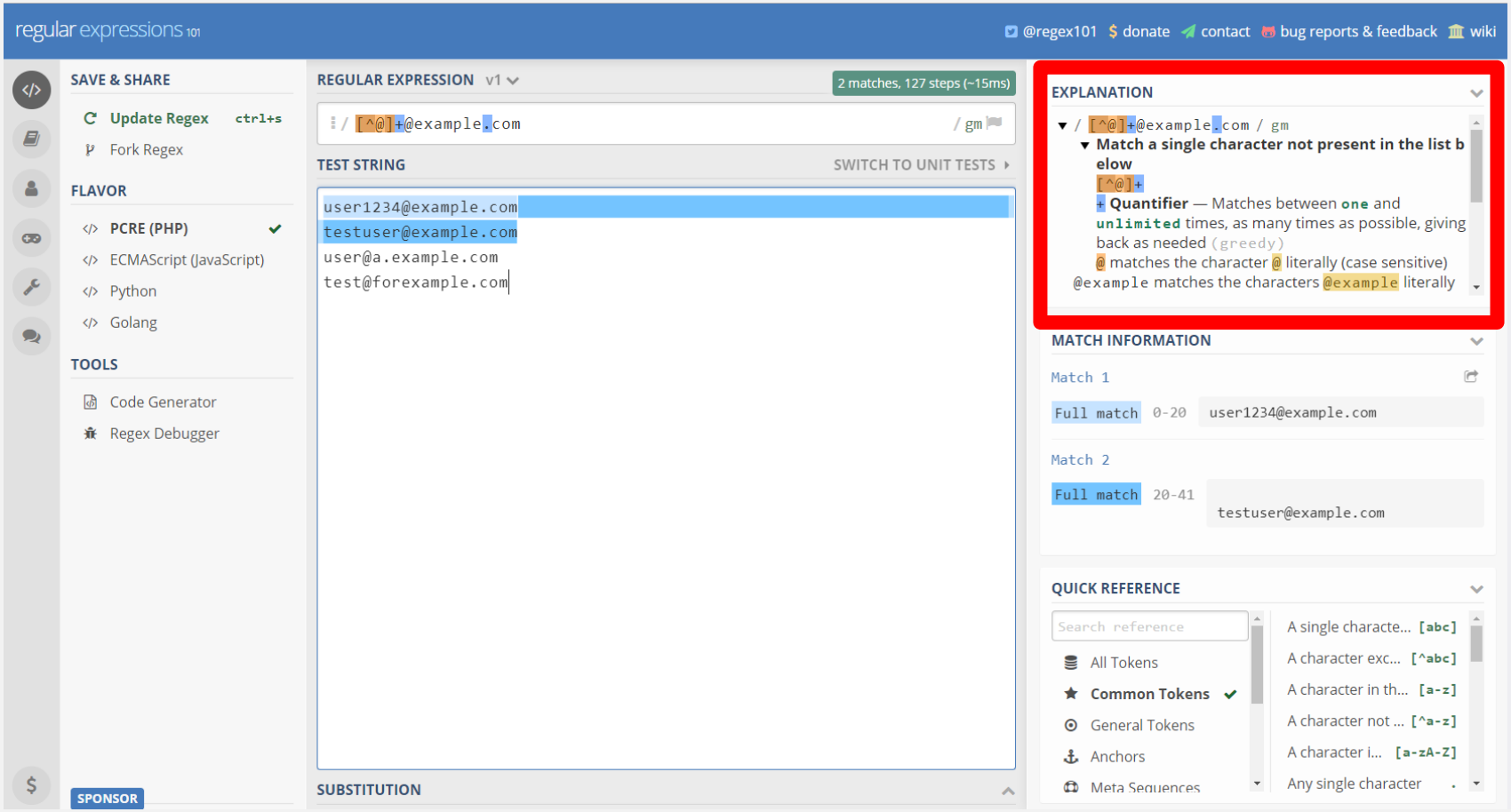
Regex101


regular expressions 101
@regex101
$ donate
contact
bug reports & feedback
wiki
SAVE & SHARE
Update Regex ctrl+s
Fork Regex
FLAVOR
PCRE (PHP)
ECMAScript (JavaScript)
Python
Golang
TOOLS
Code Generator
Regex Debugger
SPONSOR
REGULAR EXPRESSION v1
2 matches, 127 steps (~15ms)
/ [^@]+@example.com
/ gm
TEST STRING
SWITCH TO UNIT TESTS
user1234@example.com
testuser@example.com
user@a.example.com
test@forexample.com
SUBSTITUTION
EXPLANATION
/ [^@]+@example.com / gm
Match a single character not present in the list below
[^@]+
+ Quantifier — Matches between one and unlimited times, as many times as possible, giving back as needed (greedy)
@ matches the character @ literally (case sensitive)
@example matches the characters @example literally
MATCH INFORMATION
Match 1
Full match 0-20 user1234@example.com
Match 2
Full match 20-41 testuser@example.com
QUICK REFERENCE
Search reference
All Tokens
Common Tokens
General Tokens
Anchors
Meta Sequences
A single characte... [abc]
A character exc... [^abc]
A character in th... [a-z]
A character not ... [^a-z]
A character i... [a-zA-Z]
Any single character .

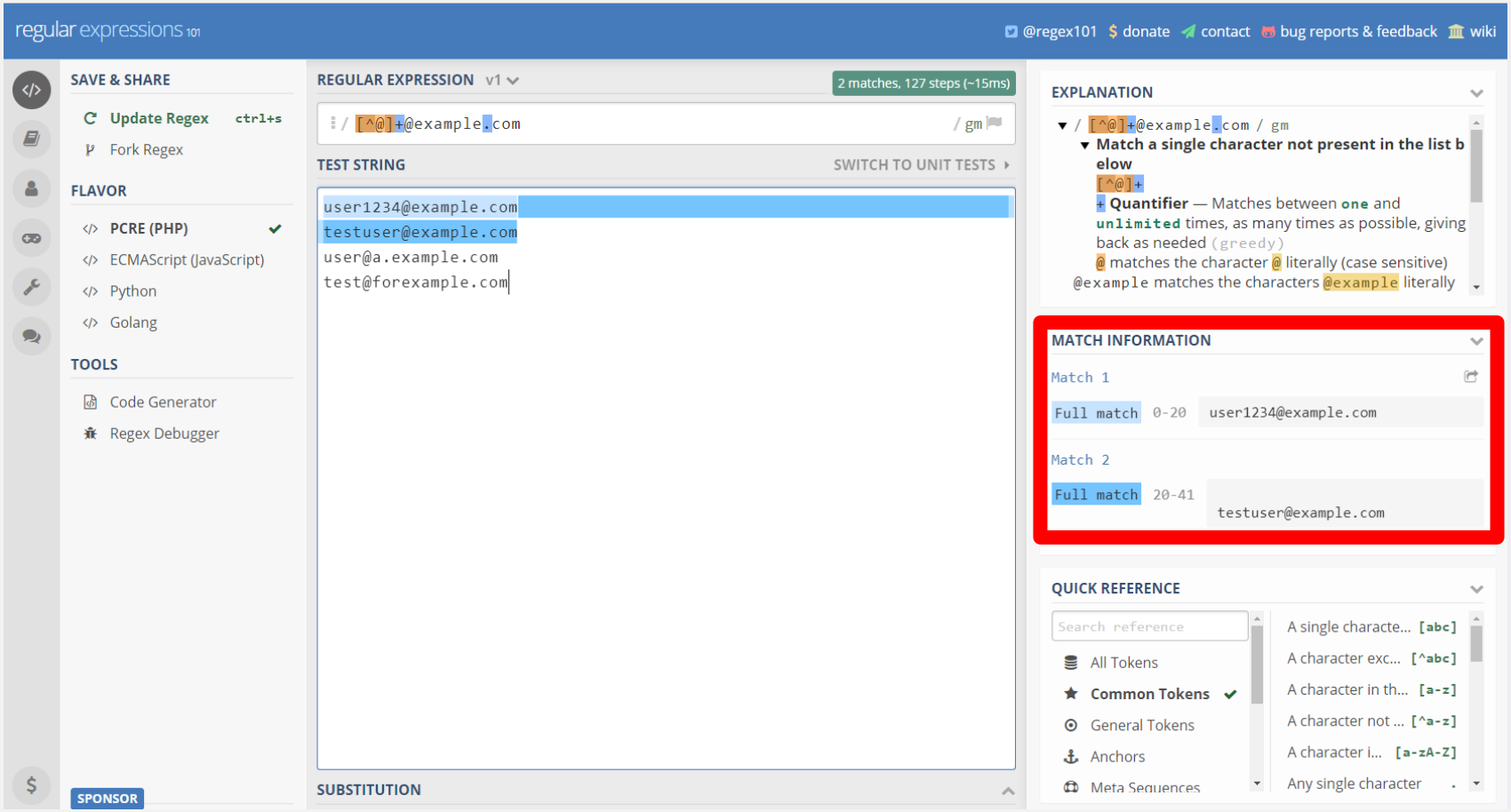
Regex101


regular expressions 101
@regex101
$ donate
contact
bug reports & feedback
wiki
SAVE & SHARE
Update Regex ctrl+s
Fork Regex
FLAVOR
PCRE (PHP)
ECMAScript (JavaScript)
Python
Golang
TOOLS
Code Generator
Regex Debugger
SPONSOR
REGULAR EXPRESSION v1
2 matches, 127 steps (~15ms)
/ [^@]+@example.com / gm
TEST STRING
SWITCH TO UNIT TESTS
user1234@example.com
testuser@example.com
user@a.example.com
test@forexample.com
SUBSTITUTION
EXPLANATION
/ [^@]+@example.com / gm
Match a single character not present in the list below
[^@]+
+ Quantifier — Matches between one and unlimited times, as many times as possible, giving back as needed (greedy)
@ matches the character @ literally (case sensitive)
@example matches the characters @example literally
MATCH INFORMATION
Match 1
Full match 0-20 user1234@example.com
Match 2
Full match 20-41 testuser@example.com
QUICK REFERENCE
Search reference
All Tokens
Common Tokens
General Tokens
Anchors
Meta Sequences
A single characte... [abc]
A character exc... [^abc]
A character in th... [a-z]
A character not ... [^a-z]
A character i... [a-zA-Z]
Any single character .

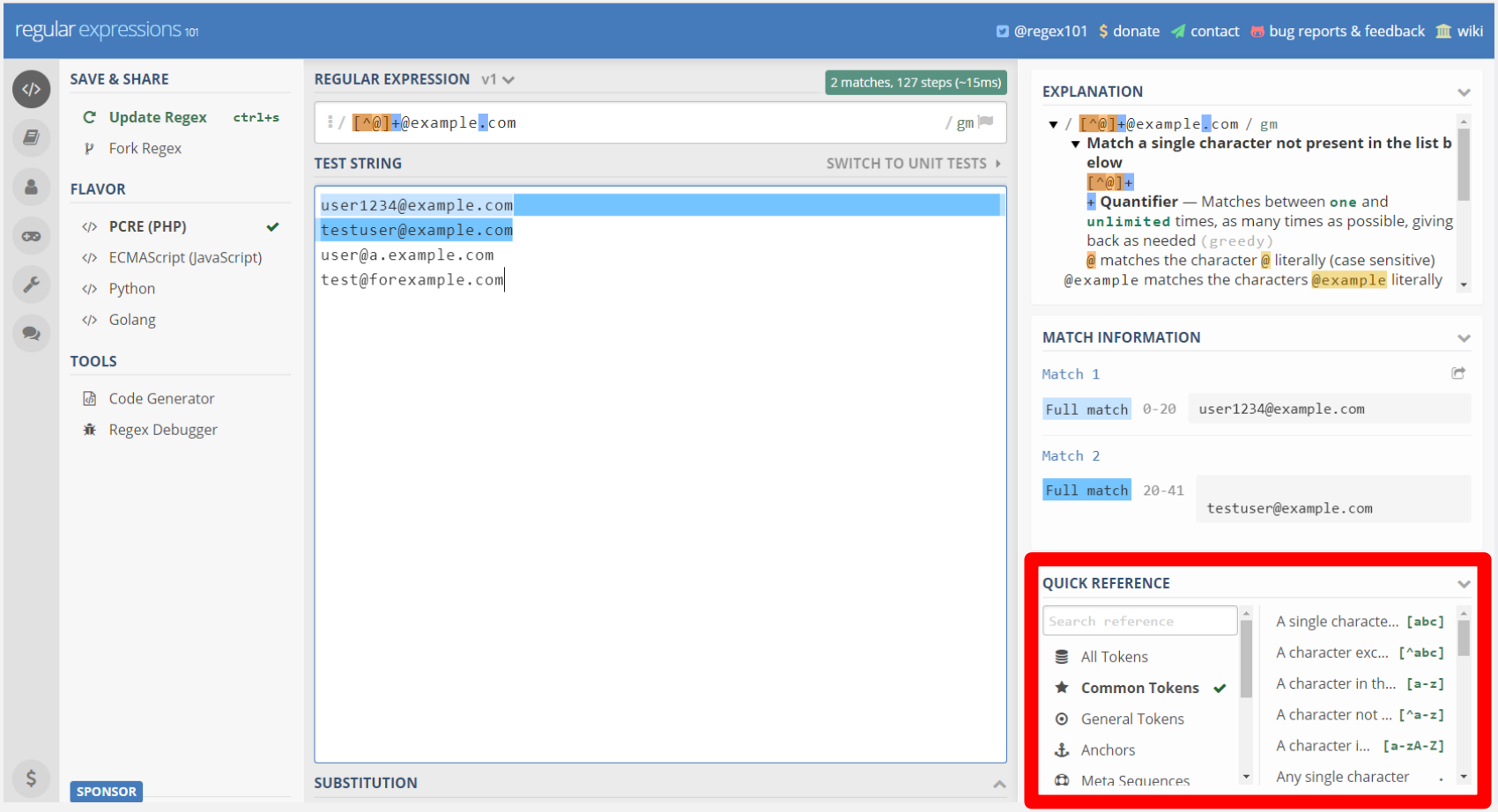
Regex101


regular expressions 101
@regex101
$ donate
contact
bug reports & feedback
wiki
SAVE & SHARE
Update Regex ctrl+s
Fork Regex
FLAVOR
PCRE (PHP)
ECMAScript (JavaScript)
Python
Golang
TOOLS
Code Generator
Regex Debugger
SPONSOR
REGULAR EXPRESSION v1
2 matches, 127 steps (~15ms)
/ [^@]+@example.com / gm
TEST STRING
SWITCH TO UNIT TESTS
user1234@example.com
testuser@example.com
user@a.example.com
test@forexample.com
SUBSTITUTION
EXPLANATION
/ [^@]+@example.com / gm
Match a single character not present in the list below
[^@]+
+ Quantifier — Matches between one and unlimited times, as many times as possible, giving back as needed (greedy)
@ matches the character @ literally (case sensitive)
@example matches the characters @example literally
MATCH INFORMATION
Match 1
Full match 0-20 user1234@example.com
Match 2
Full match 20-41 testuser@example.com
QUICK REFERENCE
Search reference
All Tokens
Common Tokens
General Tokens
Anchors
Meta Sequences
A single characte... [abc]
A character exc... [^abc]
A character in th... [a-z]
A character not ... [^a-z]
A character i... [a-zA-Z]
Any single character .

Regex101


regular expressions 101
@regex101
$ donate
contact
bug reports & feedback
wiki
SAVE & SHARE
Update Regex ctrl+s
Fork Regex
FLAVOR
PCRE (PHP)
ECMAScript (JavaScript)
Python
Golang
TOOLS
Code Generator
Regex Debugger
SPONSOR
REGULAR EXPRESSION v1
2 matches, 127 steps (~15ms)
/ [^@]+@example.com / gm
TEST STRING
SWITCH TO UNIT TESTS
user1234@example.com
testuser@example.com
user@a.example.com
test@forexample.com
SUBSTITUTION
EXPLANATION
/ [^@]+@example.com / gm
Match a single character not present in the list below
[^@]+
+ Quantifier — Matches between one and unlimited times, as many times as possible, giving back as needed (greedy)
@ matches the character @ literally (case sensitive)
@example matches the characters @example literally
MATCH INFORMATION
Match 1
Full match 0-20 user1234@example.com
Match 2
Full match 20-41 testuser@example.com
QUICK REFERENCE
Search reference
All Tokens
Common Tokens
General Tokens
Anchors
Meta Sequences
A single characte... [abc]
A character exc... [^abc]
A character in th... [a-z]
A character not ... [^a-z]
A character i... [a-zA-Z]
Any single character .

Regex101


regular expressions 101
@regex101
$ donate
contact
bug reports & feedback
wiki
SAVE & SHARE
Update Regex ctrl+s
Fork Regex
FLAVOR
PCRE (PHP)
ECMAScript (JavaScript)
Python
Golang
TOOLS
Code Generator
Regex Debugger
SPONSOR
REGULAR EXPRESSION v1
2 matches, 127 steps (~15ms)
/ [^@]+@example.com / gm
TEST STRING
SWITCH TO UNIT TESTS
user1234@example.com
testuser@example.com
user@a.example.com
test@forexample.com
SUBSTITUTION
EXPLANATION
/ [^@]+@example.com / gm
Match a single character not present in the list below
[^@]+
+ Quantifier — Matches between one and unlimited times, as many times as possible, giving back as needed (greedy)
@ matches the character @ literally (case sensitive)
@example matches the characters @example literally
MATCH INFORMATION
Match 1
Full match 0-20 user1234@example.com
Match 2
Full match 20-41 testuser@example.com
QUICK REFERENCE
Search reference
All Tokens
Common Tokens
General Tokens
Anchors
Meta Sequences
A single characte... [abc]
A character exc... [^abc]
A character in th... [a-z]
A character not ... [^a-z]
A character i... [a-zA-Z]
Any single character .

Q&A