Чтобы попрактиковать регулярки– [**https://regex101.com/**](https://regex101.com/)

**Регулярные выражения –** механизм для поиска и замены текста. Основан на использовании метасимволов. Позволяет задать шаблон, по которому будет производится поиск.

**Точный поиск строки** – самый простой вариант регулярного выражения. Ищет точно такую же строку. Регулярные выражения регистрозависимые, но многие ЯП позволяют отключить это.

Большинство символов в регулярном выражении представляет само себя за исключением специальных символов **[ ] \ / ^ $ . | ? \* + ( ) { }**

Если нужно найти спецсимвол, его нужно экранировать обратным слешом **\**.

**Любой символ .**

Метасимвол . (точка) означает один любой символ, включая цифры, спецсимволы и пробелы. Вместо символа . можно использовать [\s\S] (все пробельные и непробельные символы, включая символ новой строки).

**Символьные классы (наборы символов) [ ]**

Набор символов в квадратных скобках именуется символьным классом и позволяет указать, что на данном месте в строке может стоять один из перечисленных символов. [нл] — только «н» и «л»

[а-я] — все русские буквы в нижнем регистре от «а» до «я» (кроме «ё»)

[А-Я]    — все заглавные русские буквы

[А-Яа-яЁё]  — все русские буквы

[a-z]  — латиница мелким шрифтом

[a-zA-Z]  — все английские буквы

[0-9]  — любая цифра

[В-Ю]   — буквы от «В» до «Ю» (да, диапазон — это не только от А до Я)

[А-ГО-Р]   — буквы от «А» до «Г» и от «О» до «Р»

Диапазон всегда обозначается одним символом. [1-37] – это диапазон от 1 до 3 и число 7.

[] Именно задает «перечисление», например: [abc] и (a|b|c) по сути дают один и тот же результат, но для такого перебора возможных значений надо именно [] использовать (типа он быстрее это делает, без захвата и для этого и предназначен).

Для удобства значения можно разделять запятыми [a-d,f,u-z].

**Метасимволы**

|  |  |  |
| --- | --- | --- |
| **Символ** | **Возможный эквивалент** | **Соответствие** |
| **\d** | [0-9] | Цифровой символ |
| **\D** | [^0-9] | Нецифровой символ |
| **\s** | [ \f\n\r\t\v] | Пробельный символ |
| **\S** | [^ \f\n\r\t\v] | Непробельный символ  **Пример:** Выражение вида ^\S.\* или ^[^ \f\n\r\t\v].\* будет находить строки, начинающиеся с непробельного символа |
| **\w** | [A-Za-z0-9\_] | Буквенный или цифровой символ или знак подчёркивания; буквы ограничены латиницей  **Пример:** Выражение вида \w+ будет находить и выделять отдельные слова |
| **\W** | [^A-Za-z0-9\_] | Любой символ, кроме буквенного или цифрового символа или знака подчёркивания |
| **\r** |  | Возврат каретки |
| **\n** |  | Перевод строки |
| **\t** |  | Табуляция |
| **\v** |  | Вертикальная табуляция |
| **\f** |  | Конец страницы |
| **\a** |  | символ оповещения, сигна, (шестнадцатиричный код 07) |
| **\e** |  | escape |
| **\b** |  | граница слова |
| **\B** |  | не является границей слова |
| **\A** |  | начало данных (независимо от многострочного режима) |
| **\Z** |  | конец данных либо позиция перед последним символом строки, в случае если это символ перевода строки (независимо от многострочного режима) |
| **\z** |  | конец данных (независимо от многострочного режима) |

Реже используются **классы символов**:

|  |  |
| --- | --- |
| **Класс символов** | **Пояснение** |
| [[:alnum:]] | Буквы или цифры: [а-яА-ЯёЁa-zA-Z0-9] |
| [[:alpha:]] | Только буквы: [а-яА-ЯёЁa-zA-Z] |
| [[:digit:]] | Только цифры: [0-9] |
| [[:graph:]] | Только отображаемые символы (пробелы, служебные знаки и т. д. не учитываются) |
| [[:print:]] | Отображаемые символы и пробелы |
| [[:space:]] | Пробельные символы [ \f\n\r\t\v] |
| [[:punct:]] | Знаки пунктуации: ! " # $ % & ' ( ) \* + , \ -. / : ; < = > ? @ [ ] ^ \_ ` { | } |
| [[:word:]] | Буквенный или цифровой символ или знак подчёркивания: [а-яА-ЯёЁa-zA-Z0-9\_] |

**Позиция внутри строки**

Следующие символы позволяют спозиционировать регулярное выражение относительно элементов текста: начала и конца строки, границ слова.

|  |  |  |  |
| --- | --- | --- | --- |
| **Представление** | **Позиция** | **Пример** | **Соответствие** |
| **^** | Начало текста (или строки при модификаторе ?m) | ^a | aaa aaa |
| **$** | Конец текста (или строки при модификаторе ?m) | a$ | aaa aaa |
| **\b** | Граница слова | a\b | aaa aaa |
| \ba | aaa aaa |
| **\B** | Не граница слова | \Ba\B | aaa aaa |
| **\G** | Предыдущий успешный поиск | \Ga | aaa aaa (поиск остановился на 4-й позиции — там, где не нашлось a) |

**Специальные символы**

**\n** — перевод строки

**\r** — возврат каретки

**Обозначение группы ( )**

**Группа** – пронумерованная часть регулярного выражения.

Круглые скобки используются для определения области действия и приоритета операций. Шаблон внутри группы обрабатывается как единое целое и может быть квантифицирован. Например, выражение (тр[ау]м-?)\* найдёт последовательность вида трам-трам-трумтрам-трум-трамтрум.

Подстроки, найденные по шаблону внутри группы, **сохраняются** в отдельной области памяти и **получают номер**, начиная с 1. Впоследствии в пределах данного регулярного выражения можно использовать обозначения **от \1 до \9** для проверки на совпадение с ранее найденной подстрокой. Это так называемая **ссылка назад (обратная ссылка).**

Группа /0 это все выражение в целом.

Группы можно **именовать**

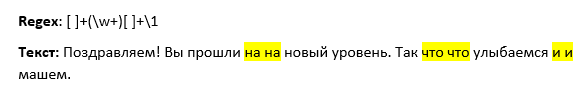
(?<gname>la|lo)-\k<gname>

**Атомарная группировка** похожа по действию с ревнивой квантификацией.

/a(?>bc|b|x)cc/

Например такое выражение найдя совпадение с bc не будет выполняться проверка альтернатив, сразу перекинется на поиск строки “cc”. Строка abcc не будет соответствовать вышеприведенному регулярному выражению.

Например найти опечатки-повторы:



**Перечисление |**

Вертикальная черта разделяет допустимые варианты (Работает как или). Например, gray|grey соответствует gray или grey. Следует помнить, что перебор вариантов выполняется слева направо, как они указаны.

Если требуется указать перечень вариантов внутри более сложного регулярного выражения, то его нужно заключить в группу. Например, gray|grey или gr(a|e)y описывают строку gray или grey. В случае с односимвольными альтернативами предпочтителен вариант gr[ae]y, так как сравнение с символьным классом выполняется проще, чем обработка группы с проверкой на все её возможные модификаторы и генерацией обратной связи.

**Квантификация (поиск последовательностей)**

Квантификатор после символа, символьного класса или группы определяет, *сколько раз предшествующее выражение может встречаться*. Следует учитывать, что квантификатор может относиться более чем к одному символу в регулярном выражении, только если это символьный класс или группа.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Представление** | **Число повторений** | **Эквивалент** | **Пример** | **Соответствие** |
| ? | Ноль или одно | {0,1} | colou?r | color, colour |
| \* | Ноль или более | {0,} | colou\*r | color, colour, colouur и т. д. |
| + | Одно или более | {1,} | colou+r | colour, colouur и т. д. (но не color) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Представление** | **Число**  **повторений** | **Пример** | **Соответствие** |
| {*n*} | Ровно *n* раз | colou{3}r | colouuur |
| {*m*,*n*} | От *m* до *n*  включительно | colou{2,4}r | colouur, colouuur, colouuuur |
| {*m*,} | Не менее *m* | colou{2,}r | colouur, colouuur, colouuuur и т. д. |
| {,*n*} | Не более *n* | colou{,3}r | color, colour, colouur, colouuur |

Часто используется последовательность .\* для обозначения любого количества любых символов между двумя частями регулярного выражения.

Символьные классы в сочетании с квантификаторами позволяют устанавливать соответствия с реальными текстами. Например, столбцами цифр, телефонами, почтовыми адресами и др.

Если символы **{ }** не образуют квантификатора, их специальное значение игнорируется.

**Жадная и ленивая квантификация**

В некоторых реализациях квантификаторам в регулярных выражениях соответствует максимально длинная строка из возможных (квантификаторы являются ***жадными***). Это может оказаться значительной проблемой. Например, часто ожидают, что выражение (<.\*>) найдёт в тексте теги HTML. Однако если в тексте есть более одного HTML-тега, то этому выражению соответствует целиком строка, содержащая множество тегов.

**Пример использования жадных и ленивых выражений**

Выражение (<.\*>) соответствует строке, содержащей несколько тегов [HTML](https://ru.wikipedia.org/wiki/HTML)-разметки, целиком.

**<p><b>Википедия</b> — свободная энциклопедия, в которой <i>каждый</i> может изменить или дополнить любую статью.</p>**

Чтобы выделить отдельные теги, можно применить ленивую версию этого выражения: (<.\*?>) Ей соответствует не вся показанная выше строка, а отдельные теги (выделены цветом):

<p><b>Википедия</b> — свободная энциклопедия, в которой <i>каждый</i> может изменить или дополнить любую статью.</p>

Эту проблему можно решить двумя способами.

1. Учитывать символы, *не* соответствующие желаемому образцу (<[^>]\*> для вышеописанного случая).
2. Определить квантификатор как ***нежадный*** (***ленивый***, [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *lazy*) — большинство реализаций позволяют это сделать, добавив после него знак вопроса.

Использование ленивых квантификаторов может повлечь за собой обратную проблему, когда выражению соответствует слишком короткая, в частности, пустая строка.

|  |  |
| --- | --- |
| **Жадный** | **Ленивый** |
| \* | \*? |
| + | +? |
| {*n*,} | {*n*,}? |

Также общей проблемой как жадных, так и ленивых выражений являются точки возврата для перебора вариантов выражения. Точки ставятся после каждой итерации квантификатора. Если интерпретатор не нашёл соответствия после квантификатора, то он начинает возвращаться по всем установленным точкам, пересчитывая оттуда выражение по-другому.

**Ревнивая (сверхжадная) квантификация** не только старается найти максимально длинный вариант, но еще и не позволяет алгоритму возвращаться к предыдущим шагам поиска. Использование ревнивых квантификаторов увеличивает скорость поиска, особенно в тех случаях, когда строка не соответствует регулярному выражению.

Кроме того, ревнивые квантификаторы могут быть использованы для исключения нежелательных совпадений.

*один раз что-то «схватив», они никогда не откатываются назад, они не «отдают» кусочки схваченного ими следующим частям регекса.*

|  |  |
| --- | --- |
| **Жадный** | **Ревнивый** |
| \* | \*+ |
| ? | ?+ |
| + | ++ |
| {n,} | {n,}+ |

|  |  |
| --- | --- |
| **Пример** | **Соответствие** |
| ab(xa)\*+a | abxa**a**bxaa; но не abxaabxaa, так как буква **a** уже занята |

**Модификаторы –** определенные режимы поиска соответствий. Модификаторы действуют с момента вхождения и до конца регулярного выражения или противоположного модификатора.

|  |  |  |
| --- | --- | --- |
| **Синтаксис** | **Описание** | |
| (?i) | Включает | нечувствительность выражения к регистру символов ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *case insensitivity*) |
| (?-i) | Выключает |
| (?s) | Включает | Чтобы точка учитывала перенос строк |
| (?-s) | Выключает |
| (?m) | Символы ^ и $ вызывают соответствие только | после и до символов новой строки (искать в каждой строке) |
| (?-m) | с началом и концом текста |
| (?x) | Включает | режим без учёта пробелов между частями регулярного выражения и позволяет использовать # для комментариев |
| (?-x) | Выключает |

**Регулярное выражение китайских иероглифов** (пиздец какой-то вопрос от черкасова)

^[\u2E80-\u9FFF]+$ Соответствие всем восточноазиатским языкам

^[\u4E00-\u9FFF]+$ Матч упрощенный и традиционный

^[\u4E00-\u9FA5]+$ Упрощенный матч

Во многих реализациях можно использовать так называемые **Unicode Scripts**:

\p{Cyrillic}

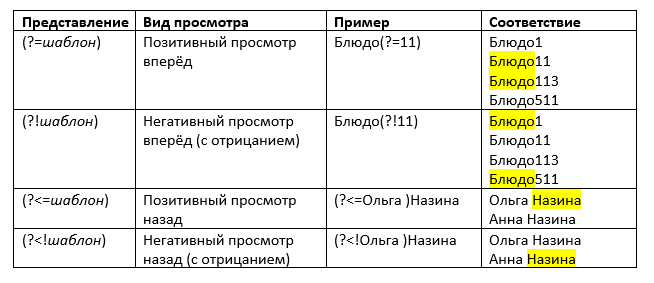
\p{Latin}

\p{Han} – Китайские

Либо **UnicodeBlock,** которые соответствуют промежутку в таблице Unicode.

**ПРОСМОТР ВПЕРЕД И НАЗАД**

Иногда нужно включить в выборку подстроку, рядом с которой содержится определенный текст.



Вопросы на собесах

Рекомендуется еще раз посмотреть на задачу по валидации значения цвета (строго либо #xxx, либо #xxxxxx)

Разобраться как работать с национальными символами (иероглифы), лучше разобраться с отличием жадного и сверхжадного квантификатора на примерах

повторить жадный и ленивый поиск, группы

В теории всё хорошо, но на практике были проблемы c квантификацией скобочных групп

Местами путался по символьным классам, но после замечаний сам исправлялся, флаг g

Что спрашивали у меня

1. Рассказать что такое регулярки и зачем они вообще нам нужны
2. Написать регулярку для задания цвета (либо 3 символа либо 6)

#xxx

#xxxxxx

Я написала так - /^#([A-Fa-f\d]{3}|[A-Fa-f\d]{6})$ (сказали правильно)

1. Дали список и сказали объяснить что это и зачем (выше написано)

|  |
| --- |
| **\d** |
| **\D** |
| **\s** |
| **\S** |
| **\w** |
| **\W** |
| **^ это указывает на начало строки, но черкасов спросил где еще используется. Ответ – для отрицания** |
| **$** |
|  |

1. Обязательно спросят про ленивую, жадную и сверхжадную квантификацию, как они пишутся, как они работают и тд. Я приводила пример сразу регуляркой и разными квантификаторами, говорила что они найдут. Это любимый вопрос черкасова в этой теме как я поняла
2. Что такое скобочная группа и зачем она нужна (я сказала про область действия квантификатора и для того чтобы использовать перечисление, но еще суть в том что регулярку можно на логические группы делить и легко парсить потом)
3. Спросили про все модификаторы