Альтернация (или) |

Альтернация – термин в регулярных выражениях, которому в русском языке соответствует слово «ИЛИ».

В регулярных выражениях она обозначается символом вертикальной черты `|`.

Например, нам нужно найти языки программирования: HTML, PHP, Java и JavaScript.

Соответствующее регулярное выражение: `html|php|java(script)?`.

Пример использования:

```
1  let regexp = /html|css|java(script)?/gi;
2
3  let str = "Сначала появился язык Java, затем HTML, пото
4
5  alert( str.match(regexp) ); // Java,HTML,JavaScript
```

Мы уже видели нечто подобное – квадратные скобки. Они позволяют выбирать между несколькими символами, например `gr[ae]y` найдёт `gray`, либо `grey`.

Квадратные скобки работают только с символами или наборами символов. Альтернация мощнее, она работает с любыми выражениями. Регулярное выражение `A|B|C` обозначает поиск одного из выражений: `A`, `B` или `C`.

Например:

- `gr(a|e)y` означает точно то же, что и `gr[ae]y`.
- `gra|ey` означает `gra` или `ey`.

Чтобы применить альтернацию только к части шаблона, можно заключить её в скобки:

- `Люблю HTML|CSS` найдёт `Люблю HTML` или `CSS`.
- `Люблю (HTML|CSS)` найдёт `Люблю HTML` или `Люблю CSS`.

Пример: шаблон для времени

В предыдущих главах было задание написать регулярное выражение для поиска времени в формате `чч:мм`, например `12:00`. Но шаблон `\d\d:\d\d` недостаточно точный. Он принимает `25:99` за время (99 секунд подходят под шаблон, но так не должно быть).

Как сделать лучше?

Мы можем применить более тщательное сравнение. Во-первых, часы:

- Если первая цифра `0` или `1`, тогда следующая цифра может быть любой: `[01]\d`.
- Или если первая цифра `2`, тогда следующая должна быть от 0 до 3: `2[0-3]`.
- (другой первой цифры быть не может)

В виде регулярного выражения оба варианта для часов можно записать при помощи альтернации: `[01]\d|2[0-3]`.

Далее, минуты должны быть от `00` до `59`. На языке регулярных выражений это означает `[0-5]\d`: первая цифра `0-5`, а за ней любая.

Давайте соединим часы и минуты в одно выражение, получится так: `[01]\d|2[0-3]:[0-5]\d`.

Почти готово, но есть проблема. После такого соединения альтернация `|` оказалась между `[01]\d` и `2[0-3]:[0-5]\d`.

То есть, минуты добавились ко второму варианту альтернации, вот более наглядно:

```
[01]\d  |  2[0-3]:[0-5]\d
```

Такой шаблон будет искать `[01]\d` или `2[0-3]:[0-5]\d`.

Но это неверно. Нам нужно, чтобы альтернация использовалась только внутри части регулярного выражения, относящейся к часам, чтобы разрешать `[01]\d` ИЛИ `2[0-3]`. Для этого обернём «часы» в скобки: `([01]\d|2[0-3]):[0-5]\d`.

Пример работы:

```
let regexp = /([01]\d|2[0-3]):[0-5]\d/g;

alert("00:00 10:10 23:59 25:99 1:2".match(regexp)); //
```

Задачи

Найдите языки программирования

Существует много языков программирования, например, Java, JavaScript, PHP, C, C++.

Напишите регулярное выражение, которое найдёт их все в строке `Java JavaScript PHP C++ C`:

```
let regexp = /ваше регулярное выражение/флаги;

alert("Java JavaScript PHP C++ C".match(regexp)); // Ja
```

решение

Найдите пары BB-кодов

BB-код имеет вид `[tag]...[/tag]`, где `tag` – это один из: `b`, `url` или `quote`.

Например:

```
[b]текст[/b]
[url]http://ya.ru[/url]
```

BB-коды могут быть вложенными. Но сам в себя тег не может быть вложен, например:

```
Возможно:
[url] [b]http://ya.ru[/b] [/url]
[quote] [b]текст[/b] [/quote]

Не может быть:
[b][b]текст[/b][/b]
```

Теги могут содержать переносы строк, это допустимо:

```
[quote]
  [b]текст[/b]
[/quote]
```

Создайте регулярное выражение для поиска всех BB-кодов и их содержимого.

Например:

```
let regexp = /ваше регулярное выражение/флаги;

let str = "..[url]http://ya.ru[/url]..";
alert( str.match(regexp) ); // [url]http://ya.ru[/url]
```

Если теги вложены, то нужно искать самый внешний тег (при желании можно продолжить поиск в его содержимом):

```
let regexp = /ваше регулярное выражение/флаги;

let str = "..[url][b]http://ya.ru[/b][/url]..";
alert( str.match(regexp) ); // [url][b]http://ya.ru[/b]
```

решение

Найдите строки в кавычках

Создайте регулярное выражение для поиска строк в двойных кавычках `"..."`.

Важно, что строки должны поддерживать экранирование с помощью обратного слеша, по аналогии со строками JavaScript. Например, кавычки могут быть вставлены как `\"`, новая строка как `\n`, а сам обратный слеш как `\\`.

```
let str = "Как вот \"здесь\".";
```

В частности, обратите внимание: двойная кавычка после обратного слеша `\"` не оканчивает строку.

Поэтому мы должны искать от одной кавычки до другой, игнорируя встречающиеся экранированные кавычки.

В этом и состоит основная сложность задачи, которая без этого условия была бы элементарной.

Примеры подходящих строк:

```
.. "test me" ..
.. "Скажи \"Привет\"!" ... (строка с экранированными ка
.. "\\" ..  (внутри двойной слеш)
.. "\\ \"" ..  (внутри двойной слеш и экранированная ка
```

В JavaScript приходится удваивать обратные слеши, чтобы добавлять их в строку, как здесь:

```
let str = ' .. "test me" .. "Скажи \\"Привет\\"!" ..

// эта строка в памяти:
alert(str); //  .. "test me" .. "Скажи \"Привет\"!" ..
```

решение

Найдите весь тег

Напишите регулярное выражение, которое ищет тег `<style...>`. Оно должно искать весь тег: он может как не иметь атрибутов `<style>`, так и иметь несколько `<style type="..." id="...">`.

...Но регулярное выражение не должно находить `<styler>`!

Например:

```
let regexp = /ваше регулярное выражение/g;

alert( '<style> <styler> <style test="...">'.match(rege
```

решение

Комментарии

перед тем как писать...