Объект RegExp. Метасимволы

Введение

До сих пор мы составляли простейшие шаблоны, такие шаблоны, которые не нуждаются в каких-либо специальных символах. Для выполнения более сложных задач нам потребуются **специальные символы** (метасимволы).

Несколько слов для тех, кто не знаком с парадигмой метасимволов. Запомните:

- Работать будем с теми же **методами объекта RegExp** (match, search, replace и др.).
- **Шаблон RegExp** будем описывать не строковыми литералами, а **символами**.

> ***Примечание***. Напомню, некоторые методы регулярного выражения вызываются на объекте **String**, некоторые на объекте **RegExp**.
>
> - **String**.метод
> - **RegExp**.метод

Следующий пример не для построчного разбора кода, а для понимания **алгоритма** выполняемой задачи.

example_1. Первый поиск с использованием метасимволов

```
<script>
    // определение строки для поиска
    let str = "Языки PHP и JavaScript монстры Fullstack
    разработки";
    // 1-й шаблон поиска
    // ищем слово JavaScript в любом регистре
    let reg1 = /JavaScript/gi;
    // 2-й шаблон поиска
    // ищем любые английские слова, где:
    // - допустимые буквы от 'a' до 'z'
    // - количество букв от 3 до 15
    let reg2 = /[a-z]{3,15}/gi;
    // вывод результата поиска:
    // ['JavaScript']
    console.log(str.match(reg1));
    // вывод результата поиска:
    // ['PHP', 'JavaScript', 'Fullstack']
    console.log(str.match(reg2));
    // метасимволы читаются любыми методами
</script>
```

Метасимволы задают тип символов искомой строки, способ окружения искомой строки в тексте, а также количество символов отдельного типа в просматриваемом тексте.

Метасимволы

Для указания регулярных выражений используются метасимволы.

Метасимволы — это **символы**, которые **интерпретируются особым образом** механизмом RegEx. Вот список **метасимволов**:

. | [] () \ * + ? {} ^ $

> ***Важно***. Постарайтесь вникнуть в логику **сопоставления** шаблонов, использующих **метасимволы**. Умение писать шаблоны это не просто, и в случае с метасимволами регулярных выражений навык обретается только практикой. Экспериментируйте. И помните, у самурая нет цели... ;)

Поэтому метасимволы можно разделить на три группы:

- Метасимволы поиска совпадений.
- Количественные метасимволы.
- Метасимволы позиционирования.

Метасимволы поиска совпадений

Метасимволы, определяющие совпадения, начнем с самого универсального.

. - Точка

Точка "." соответствует **любому одиночному символу** (кроме новой строки '\n').

Шаблон	Строка	Количество совпадений
. .	a	`console.log("a".match(/../g)); // null`
	ab	`console.log("ab".match(/../g)); // 1`
	abc	`console.log("abc".match(/../g)); // 1`

	abcd	`console.log("abcd".match(/../g)); // 2`

Шаблон **..** будет соответствовать, если сопоставляемая строка содержит два или более символов.

example_2. Универсальный метасимвол - точка

```
<script>
    console.log("a".match(/../g)); // null
    console.log("ab".match(/../g)); // ['ab']
    console.log("abc".match(/../g)); // ['ab']
    console.log("abcd".match(/../g)); // ['ab','cd']
</script>
```

| - Альтернатива (или)

Альтернатива (альтернация, вертикальная черта) "|" – термин в регулярных выражениях, которому в русском языке соответствует слово «ИЛИ» (логический оператор OR).

Например: gr(a|e)y означает точно то же, что и gr[ae]y.

Шаблон	Строка	Количество совпадений
a|b	abcde	`console.log("abcde".match(/a|b/g)); // 2`
	cde	`console.log("cde".match(/a|b/g)); // null`
	acde	`console.log("acde".match(/a|b/g)); // 1`
	abcdea	`console.log("abcdea".match(/a|b/g)); // 3`

Шаблон **a|b** будет сопоставлен с любой строкой, содержащей либо, "a" либо "b". Чтобы применить альтернацию только к **части шаблона**, можно заключить её в скобки.

example_3. Метасимвол - альтернатива

```
<script>
    console.log("abcde".match(/a|b/g)); // ['a','b']
    console.log("cde".match(/a|b/g)); // null
    console.log("acde".match(/a|b/g)); // ['a']
    console.log("abcdea".match(/a|b/g)); // ['a','b','a']
</script>
```

[] – Скобочные выражения (квадратные скобки)

Квадратные скобки "[]" указывают на **символ** из **набора**, который вы хотите сопоставить.

Шаблон	Строка	Количество совпадений
[abc]	a	`console.log("a".match(/[abc]/g)); // 1`
	ab	`console.log("ab".match(/[abc]/g)); // 2`
	abc de ba	`console.log("abc de ba".match(/[abc]/g)); // 5`
	hello	`console.log("hello".match(/[abc]/g)); // null`

Шаблон [**abc**] будет соответствовать, если сопоставляемая строка содержит любой из символов "a", "b" или "c", **но только один из них**.

> ***Важно***. Если для квадратных скобок шаблона не указан специальный количественный метасимвол (**квантификатор**) всегда подразумевается **только один** символ из предлагаемого набора.

Можно указать **диапазон символов**, используя дефис "-" в квадратных скобках.

- [a-e] то же самое, что и [abcde].
- [1-4] то же самое, что и [1234].
- [0-39] то же самое, что и [01239].

Вы можете инвертировать набор символов, используя символ "^" в начале квадратной скобки.

- [^abc] означает любой символ, кроме a или b или c.
- [^0-9] означает любой нецифровой символ.

Помните, что внутри скобочных выражений все специальные символы (включая обратную косую черту \) **теряют своё служебное значение**, поэтому экранировать их не нужно (об экранировании дальше)

example_4. Скобочные выражения

```
<script>
    console.log("a".match(/[abc]/g)); // ['a']
    console.log("ab".match(/[abc]/g)); // ['a','b']
    console.log("abc de ba".match(/[abc]/g)); //
    ['a','b','c','b','a']
    console.log("hello".match(/[abc]/g)); // null
</script>
```

() - Скобочные группы

Круглые скобки "()" используются для **группировки подшаблонов**.

Так, например, (a|b|c)xz соответствует любой строке, которая соответствует либо "a" или "b" или "c" с последующим "xz".

Шаблон	Строка	Количество совпадений
(a|b|c)de	ab de	`// null` `console.log("ab de".match(/(a|b|c)de/g));`
	ade	`// 1` `console.log("ade".match(/(a|b|c)de/g));`
	acde	`// 1` `console.log("acde".match(/(a|b|c)de/g));`

	ade hibde	`// 2` `console.log("ade hibde".match(/(a|b|c)de/g));`

Шаблон (**a**|**b**|**c**)**de** будет соответствовать одному из трех символов "a" или "b" или "c" за которым следуют символы "de".

example_5. Скобочные группы

```
<script>
    console.log("ab de".match(/(a|b|c)de/g)); // null
    console.log("ade".match(/(a|b|c)de/g)); // ['ade']
    console.log("acde".match(/(a|b|c)de/g)); // ['cde']
    console.log("ade hibde".match(/(a|b|c)de/g)); //
    ['ade','bde']
</script>
```

Выполним практический пример. В браузер выводится **строка заголовка второго уровня**. Кто-то видит, что с этой строкой не так? А с этой строкой явно не все в порядке, и я не имею ввиду смысл фразы, я бы не осмелился спорить с мэтром психоанализа.

example_6. Анализ сопоставления

```
<script>
    // строка в которой будем искать совпадение с шаблоном
    str = document.getElementsByTagName("h2")[0].innerText;
    // шаблон - поиск английских букв
    let reg = /[a-z]/ig;
    // поиск
    let arr = str.match(reg);
    // вывод позиции вхождения
    console.log(arr); // 12 - совпадений (каких?)
```

```
</script>
```

Важно. Кроме того, с помощью **скобочных групп** можно выполнить так называемый **захват переменных**.

Захват переменных

Например, **шаблон**:

- `/{([a-z-]{5,15})}/i`

сопоставляется с подстроками:

1. {image-bear}
2. {image-21}

и **в случае нахождения совпадения запоминает совпавшее значение** в переменной. В первой подстроке запомненное значение – **image-bear**. Во второй подстроке совпадения не найдено (цифр в шаблоне нет).

В дальнейшем запомненное значение скобочной группы можно повторно использовать в объекте регулярного выражения. Очень часто используется, например, при поиске и замене.

Сопоставленную подстроку можно достать из **предопределённых пременных $1, $2, ..., $9** объекта RegExp.

Приведу пример использования возможности **захватывающих скобок**.

example_7. Захват переменных

```
<script>
    // забираем из документа содержимое блока div
    var str = document.body.querySelector("div").innerHTML;
    // выполняем замену с использованием захватывающих скобок
    // ([1-9]) - захват числа
```

```
    // $1 - захватываемое число
    var newstr = str.replace(/{{([1-9])}}/g, `<img src='$1.jpg'
    width='350px'>`);
    // замещаем блок новым содержимым
    document.body.querySelector("div").innerHTML = newstr;
</script>
```

Захват в регулярных выражениях используется часто, так что потренируемся дополнительно в следующих примерах и самостоятельных работах.

Захват групп ведёт к проседанию производительности. Если вам не нужно повторно ссылаться на захваченную подстроку, лучше использовать скобки **без захвата** – **(?:)**.

Важно.

- **(abc)** — Это сгруппирует несколько символов вместе и **запомнит подстроку**, соответствующую им, для последующего использования. Это называется **скобочной группой**.
- **(?:abc)** — Это также объединяет несколько символов вместе, но не запоминает совпадение. Это **незапоминаемая скобочная группа**.

Количественные метасимволы

Ниже представлены варианты указания **количественных метасимволов**, их еще называют **квантификаторами**:

- ***** - ноль и большее количество раз.
- **+** - один и большее количество раз.
- **?** - ноль или один раз
- **{n}** - точно n раз.
- **{n,}** - n или большее количество раз.
- **{n,m}** - по крайней мере, n раз, но не более чем m раз.

* - Звездочка

Символ звездочки "*" соответствует **предыдущему символу** повторенному **0 или более раз**. Эквивалентно {0,} (кванторы будут рассмотрены далее).

Шаблон	Строка	Количество совпадений
ma*n	mn	`console.log("mn".match(/ma*n/g)); // 1`
	man	`console.log("man".match(/ma*n/g)); // 1`
	mann	`console.log("mann".match(/ma*n/g)); // 1`
	main	`console.log("main".match(/ma*n/g)); // null`
	woman	`console.log("woman".match(/ma*n/g)); // 1`

Шаблон **ma*n** будет соответствовать символу "m", символу "a", повторенному 0 или более раз и символу "n".

example_8. Метасимвол - *

```
<script>
    console.log("mn".match(/ma*n/g)); // ['mn']
    console.log("man".match(/ma*n/g)); // ['man']
    console.log("mann".match(/ma*n/g)); // ['man']
    console.log("main".match(/ma*n/g)); // null
    console.log("woman".match(/ma*n/g)); // ['man']
</script>
```

+ - Плюс

Символ плюс "+" соответствует **предыдущему символу** повторенному **1 или более раз**. Эквивалентно {1,}.

Шаблон	Строка	Количество совпадений
ma+n	maan	`console.log("mn".match(/ma+n/g)); // 1`

	man	`console.log("man".match(/ma+n/g)); // 1`
	mann	`console.log("mann".match(/ma+n/g)); // 1`
	main	`console.log("main".match(/ma+n/g)); // null`
	woman	`console.log("woman".match(/ma+n/g)); // 1`

Шаблон **ma+n** будет соответствовать символу "m", символу "a", повторенному 1 или более раз и символу "n".

example_9. Метасимвол - +

```
<script>
	console.log("maan".match(/ma+n/g)); // ['maan']
	console.log("man".match(/ma+n/g)); // ['man']
	console.log("mann".match(/ma+n/g)); // ['man']
	console.log("main".match(/ma+n/g)); // null
	console.log("woman".match(/ma+n/g)); // ['man']
</script>
```

? - Вопросительный знак

Знак вопроса "?" соответствует **нулю** или **одному** вхождению предыдущего символа. То же самое, что и {0,1}. По сути, делает символ **необязательным**.

Шаблон	Строка	Количество совпадений
ma?n	mn	`console.log("mn".match(/ma?n/g)); // 1`
	man	`console.log("man".match(/ma?n/g)); // 1`
	mann	`console.log("mann".match(/ma?n/g)); // 1`
	main	`console.log("main".match(/ma?n/g)); // null`
	woman	`console.log("woman".match(/ma?n/g)); // 1`

Шаблон **ma?n** будет соответствовать символу "m", необязательному символу "a" и символу "n".

example_10. Метасимвол - ?

```
<script>
      console.log("mn".match(/ma?n/g)); // ['mn']
      console.log("man".match(/ma?n/g)); // ['man']
      console.log("mann".match(/ma?n/g)); // ['man']
      console.log("main".match(/ma?n/g)); // null
      console.log("woman".match(/ma?n/g)); // ['man']
</script>
```

{} - Фигурные скобки

Фигурные скобки "{}" обозначают **количественное вхождение** предыдущего символа, где {n,m} по крайней мере, n раз, но не более чем m раз.

Пример: {2,5} – означает, по крайней мере 2, но не больше 5 повторений предыдущего символа.

Шаблон	Строка	Количество совпадений
a{2,3}	data	`console.log("data".match(/a{2,3}/g)); // null`
	daat	`console.log("daat".match(/a{2,3}/g)); // 1`
	daaat	`console.log("daaat".match(/a{2,3}/g)); // 1`
	aa daat	`console.log("aa daat".match(/a{2,3}/g)); // 2`

Посмотрим еще один пример. RegEx [1-9]{2, 4} соответствует как минимум 2 цифрам, но не более 4-х цифр

Шаблон	Строка	Количество совпадений
[1-9]{2,4}	qw12erty	`// 1` `console.log("qw12erty".match(/[1-9]{2,4}/g));`
	11 12345	`// 2` `console.log("11 12345".match(/[1-9]{2,4}/g));`
	1 05	`// null` `console.log("1 05".match(/[1-9]{2,4}/g));`

example_11. Фигурные скобки

```
<script>
	console.log("data".match(/a{2,3}/g)); // null
	console.log("daat".match(/a{2,3}/g)); // ['aa']
	console.log("daaat".match(/a{2,3}/g)); // ['aaa']
	console.log("aa daat".match(/a{2,3}/g)); // ['aa', 'aa']
	console.log("qw12erty".match(/[1-9]{2,4}/g)); // ['12']
	console.log("11 12345".match(/[1-9]{2,4}/g)); // ['11',
	'1234']
	console.log("1 05".match(/[1-9]{2,4}/g)); // null
</script>
```

В предыдущем уроке я приводил пример позитивного кода (example_16) и обещал, что мы его оптимизируем с помощью **возможностей метасимволов**. Делаем.

example_12. Улучшаем код

```
<script>
	// забираем из документа содержимое блока div
	var str = document.body.querySelector("div").innerHTML;
	// выполняем замену
	var newstr = str.replace(/{{img([1-5])}}/g, `<img
	src='pict$1.jpg' width='350px'>`);
	// замещаем блок новым содержимым
	document.body.querySelector("div").innerHTML = newstr;
</script>
```

Сравните код двух примеров. Оцените всю мощь регулярных выражений и используемых в них метасимволов.

Метасимволы позиционирования

К метасимволам позиционирования относят символы:

- **^** - начало строки.
- **$** - конец строки.

^ - Каретка

Символ каретки "^" используется для проверки того, **начинается** ли строка с определенного символа (символ начала строки).

Шаблон	Строка	Количество совпадений
^ab	abcd	`console.log("abcd".match(/^ab/g)); // 1`
	_ab	`console.log(" ab".match(/^ab/g)); // null`
	ac	`console.log("ac".match(/^ab/g)); // null`

$ - Доллар

Символ доллара "$" используется для проверки того, **заканчивается** ли строка определенным символом (символ конца строки).

Шаблон	Строка	Количество совпадений
a$	a	`console.log("a".match(/a$/g)); // 1`
	cora	`console.log("cora".match(/a$/g)); // 1`
	form	`console.log("form".match(/a$/g)); // null`
	mam a	`console.log("mam a".match(/a$/g)); // 1`

example_13. Метасимвол конца строки

```
<script>

    console.log("a".match(/a$/g)); // ['a']
```

```
    console.log("cora".match(/a$/g)); // ['a']
    console.log("form".match(/a$/g)); // null
    console.log("mam a".match(/a$/g)); // ['a']
</script>
```

Таким образом, используя метасимволы **начала** и **конца** строки можно задать шаблон на поиск соответствия в целой строке (example_14).

Другие символы регулярных выражений

Символьные классы

- **\d** - любая цифра.
- **\D** - любой нецифровой символ.
- **\w** - буква, цифра или подчеркивание. Найдет любой словесный (латинский алфавит) символ, включая буквы, цифры и знак подчеркивания. Эквивалентно [A-Za-z0-9_].
- **\W** - любой не словесный символ. Эквивалентно [^A-Za-z0-9_].
- **\n** - символ новой строки.
- **\s** -пробел, табуляция, новая строка или перевод строки.
- **\S** - найдет любой символ, кроме пробельного.
- **\t** - табуляция.
- и некоторые др.

\ - Обратная косая черта

Допустим, мы хотим найти буквально точку. Не "любой символ", а именно точку. Чтобы использовать **специальный символ как обычный**, добавьте к нему обратную косую черту: **\.**.

Это называется **экранирование символа**. Обратная косая черта "\" используется для экранирования различных символов, включая все **метасимволы**.

Это гарантирует, что **экранированный символ не будет компилироваться по-особенному.**

Символ косой черты '/', так называемый **слэш**, не является специальным символом, но в JavaScript он используется для **открытия и закрытия регулярного выражения**:

`/...шаблон.../`, поэтому мы должны экранировать его.

Вот как выглядит поиск самой косой черты '/':

```
alert( "/".match(/\//) );
```

С другой стороны, если мы не используем короткую запись /.../, а создаём регулярное выражение, используя **new RegExp**, тогда нам не нужно экранировать косую черту.

example_14. Поиск соответствия в строке

```
<script>
    // Фамилию, Имя, Логин, Пароль храним в переменных
    // зададим шаблоны соответствия для пользовательских данных
    // используем символы начала и конца строки
    let pattern_surname = /^[a-za-яё\s]{3,20}$/ig;
    let pattern_name = /^[a-za-яё]{3,10}$/ig;
    let pattern_login = /^[a-zA-Z\d]{5,10}$/g;
    let pattern_pwd = /^[\w\d@%&\[\]]{5,15}$/g;
    // строковые данные пользователя в переменных
    let surname = "Гарсиа Фернандес";
    let name = "123Denis";
```

```
    let login = "master123й";
    let pwd = "Denis123@qw[";
    // сопоставление шаблона со строковыми данными
    // вывод результата в консоль браузера
    console.log("Фамилия: ", pattern_surname.test(surname));
    console.log("Имя: ", pattern_name.test(name));
    console.log("Логин: ", pattern_login.test(login));
    console.log("Пароль: ", pattern_pwd.test(pwd));
</script>
```

Опережающие и ретроспективные проверки — (?=) and (?<=)

d(?=r) соответствует **d**, только если после этого следует **r**, но **r** не будет входить в соответствие выражения

(?<=r)d соответствует **d**, только если перед этим есть **r**, но **r** не будет входить в соответствие выражения

example_15. Опережающий и ретроспективный поиск

```
<script>
    var text = document.querySelector("#full").innerHTML;
    // выбрать только ту строку, где в конце - \n
    var pattern = /[a-z-]{1,25}(?=\n)/ig;
    // выбрать только ту строку, где в конце - \br
    // var pattern = /[a-z-]{1,25}(?=<br>)/ig;
    console.log(text.match(pattern));
    // --------------------
    var text = document.querySelector("#frame").innerHTML;
    // выбрать только ту строку, где в начале - <span>
```

```
    var pattern = /(?<=<span>)[a-z-]{1,25}/ig;
    console.log(text.match(pattern));
    // --------------------
    var text = document.querySelector("#web").innerHTML;
    // выбрать только ту строку, где разрядов числа 7 и больше
    // т.е. больше число
    var pattern = /[a-za-я-]{3,25}(?=.*[0-9.]{7})/ig;
    console.log(text.match(pattern));
    // --------------------
</script>
```

Вы можете использовать оператор отрицания - "!".

d(?!r) соответствует d, только если после этого нет r, но r не будет входить в соответствие выражения

(?<!r)d соответствует d, только если перед этим нет r, но r не будет входить в соответствие выражения

Жадные и ленивые сопоставления

Квантификаторы (* + {}) — это **жадные** операторы, потому что они продолжают поиск соответствий, как можно глубже — **через весь текст**.

Например, выражением <.+> выделить тег ** ** не получится:

```
<div>"Однажды Арнольд сказал: "I'll be back"</div>
```

будет выделена вся строка, от первой до последней угловой скобки:

```
<div>"Однажды Арнольд сказал: "I'll be back"</div>
```

Чтобы найти только тэг div — можно использовать оператор **?**, сделав выражение **ленивым**.

example_16. Жадный поиск против ленивого

```
<script>
	var text = document.querySelector("#test").innerHTML;
	// жадный поиск
	var pattern = /<\/?.+>/g;
	res = text.match(pattern);
	console.log(res);
	// ленивый поиск
	var pattern = /<\/?.+?>/g;
	res = text.match(pattern);
	console.log(res);
</script>
```

Методы регулярных выражений можно объединять в **цепочки**.

example_17. Цепочка вызовов

```
<script>
	var text = document.querySelector("#test").innerHTML;
	// меняем тег <div> на <h1> за два вызова метода replace
	text = text.replace(/<div/, "<h1").replace(/<\/div/,
	"</h1");
	console.log(text);
	document.querySelector("#test").innerHTML = text;
</script>
```

Задание № 30

Используя возможности регулярных выражений (**метасимволы**, **захват**) напишите скрипт для **поиска** и **замены** шаблонов **подстрок** на соответствующие **изображения**.

Для выполнения задания воспользуйтесь файлами директории **раздаточного** материала.

Результат замены продемонстрирован на **рисунке** ниже.

Это моя работа, она высокооплачиваемая:

Это моя яхта, она дорогая:

Это я ...

Задание № 31

Используя возможности регулярных выражений напишите скрипт для **поиска** и **замены** шаблонов подстрок на соответствующие **изображения**. Будьте внимательны, файлы изображений и заменяемые подстроки носят **уникальный** характер (в отличии от предыдущего задания).

Для выполнения задания воспользуйтесь файлами директории **раздаточного** материала.

Результат замены продемонстрирован на **рисунке** ниже.

Программирование в среде Jav

Не защищено | js/

Обновить

Объект RegExp. Метасимволы

Стоит в поле теремок. Бежит мимо мышка-норушка. Увидела теремок, остановилась и спр
– Терем-теремок! Кто в тереме живет? Никто не отзывается. Вошла мышка в теремок и с

Прискакала к терему лягушка-квакушка и спрашивает:
– Терем-теремок! Кто в тереме живет?
– Я, мышка-норушка! А ты кто?
– А я лягушка-квакушка.
– Иди ко мне жить! Лягушка прыгнула в теремок. Стали они вдвоем жить.

Бежит мимо зайчик-побегайчик. Остановился и спрашивает:
– Терем-теремок! Кто в тереме живет?
– Я, мышка-норушка!
– Я, лягушка-квакушка!
– А ты кто?
– А я зайчик-побегайчик.
– Иди к нам жить! Заяц скок в теремок! Стали они втроем жить.

Задание № 32

Воспользуйтесь файлами директории раздаточного материала. Подключите файл **user.js** с данными пользователя. Используя регулярные выражения, напишите скрипт, проверяющий **соответствие данных** пользователя **указанному шаблону**.

Шаблоны для проверки:

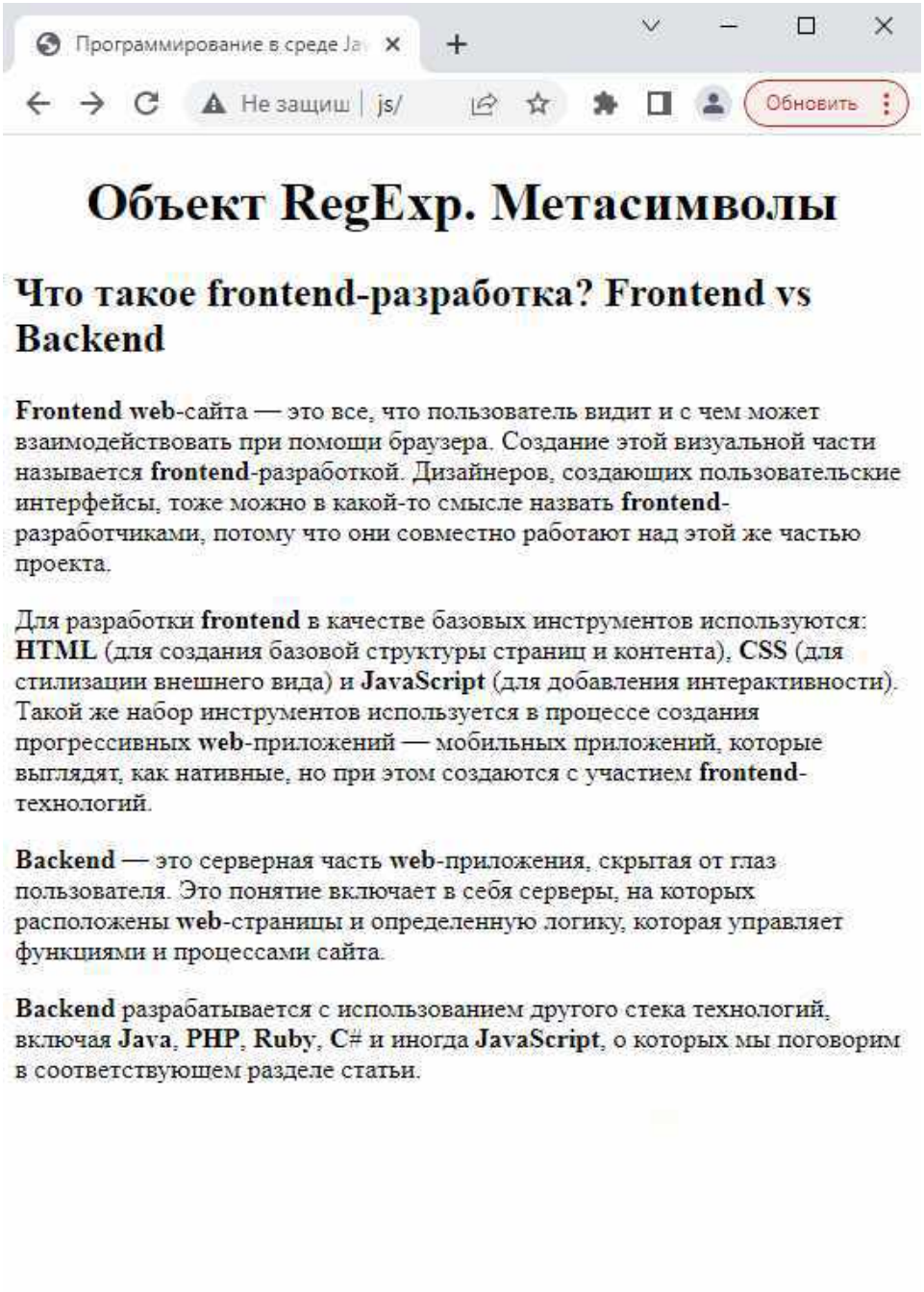
- **Имя**: символы **a-яё**, в количестве от **3** до **10**;
- **Логин**: символы **a-z0-9**, в количестве от **3** до **15**;
- **Пароль**: символы **a-z_@#$%^&*0-9-**, в количестве от **5** до **20**.

Результат проверки выведите в **консоль** браузера. В случае выявления **несоответствия** данных шаблону, выявите **причину**.

Задание № 33

В директории раздаточного материала в файле **index.html** вам предложен фрагмент статьи "**Что такое frontend-разработка? Frontend vs Backend**". Используя возможности регулярных выражений, напишите скрипт, выполняющий **выделение жирным** шрифтом всех **английских символов** статьи.

Результат работы скрипта продемонстрирован на рисунке ниже.



Задание **№3**4

Формирование объекта, создание страницы "Онлайн заказ пользователя"

В директории раздаточного материала вам предложены **исходные файлы** для выполнения практической работы. Ваша задача - используя текстовые данные файла **Заказ.pdf** создать JS-объект заказа, вывести сформированные данные о заказе пользователя в браузер.

Для выполнения практической работы, выполните следующий набор действий:

- Используя имеющиеся текстовые данные, создайте и выведите в консоль **JSON-представление** заказа.
- Используя стандартную функцию объекта JSON, преобразуйте JSON-формат в **объект JavaScript** (JS).
- Рассчитайте **конечную стоимость** каждой позиции товаров с учетом имеющегося количества.
- Рассчитайте итоговую **стоимость заказа** с учетом имеющейся скидки.
- Выведите информацию о заказе в **браузер**. Форматирование и способ внедрения кода в HTML-документ выберите по своему усмотрению.
- **Запишите расчетные значения** стоимости по каждой позиции в JS-объект заказа.
- Выведите **JS-объект** заказа в **консоль**.

Обратите внимание:

1. Править представленные данные файла **Заказ.pdf** запрещено.
2. Форматирование вывода заказа допускается выполнить по своему усмотрению.
3. На рисунке ниже представлен примерный вывод сформированных данных заказа.

Заказ пользователя: Павел Волков

Номер телефона 8(912)552-92-25
E-mail p-volkoff@mail.ru
Дата заказа 07.01.2023
Тип доставки По адресу (Москва, Белореченская улица, 31)
Скидка 10%

Артикул: RTLACB138001
Название: Носки из хлопка SUPIMA
Бренд: UNIQLO
Цвет: Синий
Сезон: Мульти
Цена: 399 ₽

(-) 3 (+) 1197 ₽

Артикул: MP002XM0984Y
Название: Брюки
Бренд: O'stin
Цвет: Черный
Сезон: Демисезон
Цена: 3799 ₽

(-) 1 (+) 3799 ₽

Артикул: MP002XM0984Y
Название: Дубленка CLAUDE
Бренд: Mango Man
Цвет: Коричневый
Сезон: Мульти
Цена: 11190 ₽

(-) 1 (+) 11190 ₽

Итого: 16186 ₽
Итого (со скидкой): 14567.4 ₽

Задание №35

Создание страницы "Семь чудес света"

В директории раздаточного материала вам предложены **исходные файлы** для выполнения практической работы.

Изучите структуру файла **seven-wonders.js** содержащего информацию о семи чудесах света в виде **массива объектов**.

Для выполнения работы выполните следующий набор действий:

- Подключите файл с данными к основному файлу **index.html**.
- Случайным образом получите **объект** одного из чудес света.
- Подготовьте к выводу в браузер **основной контент** страницы. Для формирования контента используйте следующие свойства объекта: **name**, **description**, **images**.
- Подготовьте к выводу в браузер **список ссылок** на чудеса света. Для формирования списка используйте следующие свойства объекта: **title**, **href**.
- Выполните **вставку** в документ подготовленных HTML-блоков.

Стилевое оформление страницы выполните на свое усмотрение.

Примерный вид тестовой страницы представлен в файле "**Семь чудес света.pdf**".