Объект RegExp. Основы

Введение

Регулярные выражения, обычно известные как **regex** или **RegExp**, — это **способ описания шаблонов в строковых данных**.

> ***Важно***. Регулярные выражения представляют собой специально отформатированные текстовые строки, **используемые для поиска шаблонов в тексте**.

Регулярные выражения являются одним из самых мощных инструментов, для эффективной и действенной обработки текста и манипуляций с ним.

Регулярные выражения, например, можно использовать для проверки правильности **формата** данных, введенных пользователем:

- имени;
- пароля;
- адреса электронной почты;
- номера телефона;
- поиска или замены соответствующей строки в текстовом содержимом;
- и т. д.

Эмпирическое правило заключается в том, что простые регулярные выражения просты для чтения и записи, в то время как сложные регулярные выражения сложно писать, сложно читать и сложно поддерживать/изменять.

Сложные регулярные выражения могут быстро превратиться в беспорядок, если разработчик не глубоко вникает в основы. Тем не менее, иногда регулярные выражения это единственный разумный способ выполнить какие-то манипуляции над строками, поэтому они являются очень ценным инструментом.

Они образуют небольшой отдельный язык, являющийся частью JavaScript и многих других языков, и систем. Правильное понимание регулярных выражений сделает вас более эффективным программистом.

В JavaScript регулярные выражения также являются объектами.

> ***К сведению***. Регулярные выражения – мощное средство поиска и замены в строке. Невозможно описать все возможности в одном или двух уроках, тем более что пока мы не проходили управляющие конструкции, массивы, функции. Но общий смысл и мощь регулярных выражений постараюсь показать в демонстрационных примерах и самостоятельных работах.

Создание объекта регулярного выражения

Регулярное выражение JavaScript (или RegExp) — это последовательность символов, которую мы можем использовать для эффективной работы со строками. Используя этот синтаксис, мы можем:

- **искать** текст в строке
- **заменить** подстроки в строке
- **извлекать** информацию из строки

В JavaScript **регулярное выражение** это **объект** (RegExp), который может быть определён двумя способами.

- **Литеральным** синтаксисом.
- **Конструктором объекта** регулярного выражения.

Литеральный синтаксис регулярного выражения

Синтаксис литерала использует косую черту /**pattern**/ для обертывания **шаблона** (**паттерна**) регулярного выражения.

```
let reg = /pattern/;
```

Пример:

```
// шаблон регулярного выражения – "JavaScript"
let reg = /JavaScript/;
```

example_1. Синтаксис литерала

```
<script>
    // определение строки для поиска
    let str = "Языки PHP и JavaScript монстры Fullstack
    разработки";
    // синтаксис литерала при определении шаблона поиска
    let reg = /JavaScript/;
    // вывод результата поиска
    // позиция вхождения шаблона в строку
    console.log(str.search(reg));
</script>
```

Конструктор объекта регулярного выражения

Синтаксис конструктора использует кавычки ("pattern"). Способ заключается в создании нового **объекта RegExp** с помощью ключевого слова **new**.

```
let reg = new RegExp('pattern');
```

Пример:

```
// шаблон регулярного выражения - "PHP"
let reg = new RegExp("PHP");
```

example_2. Синтаксис конструктора

```
<script>
    // определение строки поиска
    // строка содержит непечатные символы перевода строки и
    табуляции
    let str = `
    Изучаем frontend на JavaScript и
    немножко backend на PHP
    `;
    // синтаксис конструктора при определении шаблона поиска
    let reg = new RegExp("PHP");
    // вывод результата поиска
    // позиция вхождения шаблона в строку
    console.log(str.search(reg));
</script>
```

Флаги регулярных выражений

Флаги используются для изменения **поведения** метода регулярного выражения, определенного по умолчанию.

Регулярные выражения имеют несколько флагов, которые делают возможным глобальный и регистронезависимый поиск. Флаги могут использоваться **самостоятельно** или **вместе** в любом порядке, а также могут являться частью регулярного выражения.

Флаг	Описание
g	Глобальный поиск (найти все совпадения)
m	Многострочный поиск
i	Регистронезависимый поиск.

* - полный список флагов смотреть в открытых источниках

Чтобы использовать **флаги** в **шаблоне регулярного выражения** используйте следующую форму записи:

- **Синтаксис литерала**

```
let reg = /pattern/flags;
```

Пример:

```
// объект RegExp
let reg = /JavaScript/i; // флаг - "i"
```

- **Синтаксис конструктора**

```
var rexp = new RegExp('pattern', 'flags');
```

Пример:

```
// объект RegExp
let reg = new RegExp("PHP", "ig"); // комбинация флагов - "ig"
```

example_3. Определение шаблона поиска с флагом

```
<script>
    // определение строки поиска
    let str = 'Frontend разработка на JavaScript с нуля до
    гуру';
    // шаблона поиска
    let reg1 = new RegExp("javascript");
    // вывод результата поиска
```

```
    // вхождение не найдено
    console.log(str.search(reg1)); // -1
    // шаблона поиска с флагом
    let reg2 = new RegExp("javascript", "i");
    // вывод результата поиска с учетом флага -i
    // вхождение найдено в позиции 23
    console.log(str.search(reg2)); // 23
</script>
```

Методы регулярных выражений

Регулярные выражения используются:

1. в методах объекта **String**:
   - **search**
   - **match**
   - **matchAll**
   - **split**
   - **replace**
2. в методах объекта **RegExp**:
   - **test**
   - **exec**

Методы объекта String

search

```
str.search([regexp]);
```

Метод **search**() выполняет поиск сопоставления между регулярным выражением **regexp** и объектом строки **str**.

Параметр

- **regexp** – необязательный параметр. Объект регулярного выражения. Если будет передан не объект регулярного выражения, он будет **неявно преобразован в объект RegExp** через вызов конструктора **new RegExp** (regexp).

Возвращаемое значение

При успехе метод возвращает **индекс первого сопоставления** с регулярным выражением внутри строки. В противном случае метод вернёт -1.

example_4. Метод search() с неявным преобразование аргумента в RegExp

```
<script>
    // строка в которой будем искать совпадение с шаблоном
    str = document.querySelector("h2").innerText;
    // передаем не объект RegExp, а обычную строку - Homo
    // строка будет неявно преобразована автоматически
    // поиск
    let pos = str.search("Homo");
    // вывод позиции вхождения
    console.log(pos); // 37
</script>
```

example_5. Метод search() с аргументом типа RegExp

```
<script>
    // строка в которой будем искать совпадение с шаблоном
    let str = document.querySelector("pre").textContent;
```

```
    // полученная строка
    console.log(str);
    // шаблон поиска в литеральной нотации
    let reg = /Кант/;
    // поиск вхождения
    let pos = str.search(reg);
    // вывод позиции вхождения
    console.log(pos); // 213
</script>
```

match

```
str.match(regexp);
```

Метод **match**() возвращает получившиеся совпадения при сопоставлении строки **str** с регулярным выражением **regexp**.

Параметр

- **regexp** - объект регулярного выражения. Если будет передан объект **obj**, не являющийся регулярным выражением, он будет **неявно преобразован в объект RegExp** через вызов конструктора **new RegExp** (obj).

Возвращаемое значение

Объект **Array**, содержащий результаты сопоставления, или **null**, если ничего не было сопоставлено.

example_6. Метод match()

```
<script>
    // строка в которой будем искать совпадение с шаблоном
```

```
    let str = document.querySelector("pre").innerText;
    // шаблон в конструкторе
    let reg = new RegExp("левой", "i");
    // поиск вхождения
    let info = str.match(reg);
    // вывод информации о результате поиска
    // возвращается массив, но JS идентифицирует его как объект
    console.log(typeof info);  // object
    console.dir(info);
    console.dir("----------");
    console.log(info[0]);
    console.log(info['index']);
    console.log(info['input']);
</script>
```

example_7. Метод match() с флагом g

```
<script>
    // строка в которой будем искать совпадение с шаблоном
    let str = document.getElementsByTagName("h3")[0].innerText;
    // шаблон в литеральной нотации
    // let reg = /submarine/;
    // флаг -g позволяет найти все совпадения
    let reg = /submarine/g;
    // поиск вхождений
    let arr = str.match(reg);
    // вывод массива совпадений
```

```
    console.dir(arr);
</script>
```

matchAll

```
str.matchAll(regexp);
```

Метод **matchAll**() возвращает **итератор** по всем результатам при сопоставлении строки **str** с регулярным выражением **regexp**.

Параметр

- **regexp** – объект регулярного выражения. Если передано значение, не являющееся объектом регулярного выражения, оно **неявно преобразуется в RegExp** используя **new RegExp**(obj).

Возвращаемое значение

Возвращается **iterator**. Итератор можно использовать с **циклическими конструкциями** for...of, Array.from() и т.д. (циклы изучаем в отдельной теме).

example_8. Метод matchAll()

```
<script>
    // строка в которой будем искать совпадение с шаблоном
    let text = document.querySelector("pre").innerText;
    // глобальный шаблон поиска
    let reg = new RegExp("аптека", "ig");
    // возвращает специальный итератор
    let arr = text.matchAll(reg);
    // с помощью цикла выводим возвращаемый объект
    for (el of arr) {
        // протестируйте вывод, там много интересного
```

```
            // console.log(el);
            // в консоль
            console.log(el[0]);
            // в браузер
            document.write(el[0], "<br/>");
        }
        // RegExpStringIterator
        console.dir(arr);
</script>
```

split

```
str.split([separator[, limit]]);
```

Метод **split**() разбивает строку **str** на массив строк путём разделения строки указанной подстрокой **separator**.

Параметры

- **separator** – необязательный параметр. Указывает символы, используемые в качестве **разделителя** внутри строки. Параметр separator может быть как **строкой**, так и **регулярным выражением**.
    - Если разделитель **separator** найден, **он удаляется из строки**, а **подстроки возвращаются в массиве**.
    - Если параметр опущен, возвращённый массив будет содержать **один элемент со всей строкой**.
    - Если параметр равен пустой строке, строка **str** будет преобразована в **массив символов**.
- **limit** – необязательный параметр. Целое число, определяющее ограничение на **количество найденных подстрок**. Метод **split**() всё равно разделяет строку на каждом сопоставлении с разделителем **separator**, но обрезает возвращаемый массив так, чтобы он содержал **не более limit элементов**.

Возвращаемое значение

Возвращает новый **массив**.

example_9. Метод split()

```
<script>
	// строка в которой будем искать совпадение с шаблоном
	let str = document.querySelector("ul").innerText;
	// вывод строки в консоль
	console.log(str);
	// шаблон для разделения
	let reg = new RegExp("\n");
	// разбиваем строку по шаблону в массив
	let arr = str.split(reg);
	// вывод массива
	console.log(arr); // Array
	// тестируем доступ к массиву
	console.log(arr[3]);
</script>
```

В следующем примере поработаем со строкой формата **CSV**. Формат может использоваться при **асинхронном взаимодействии** клиент-сервер. На протяжении всего курса **JSON** и **CSV** будут встречаться многократно.

example_10. Метод split() и CSV формат

```
<script>
	// строка, в которой будем искать совпадение с шаблоном
	let text =
	`1;'Aerosmith';'aerosmith';'США';'1970';'хард-
```

```
    рок';'assets/teams/aerosmith.jpg'
    2;'Pink Floyd';'pink-
    floyd';'Великобритания';'1965';'психоделический-
    рок';'assets/teams/pink-floyd.jpg'
    3;'The Beatles';'the-beatles';'Великобритания';'1960';'рок-
    н-ролл';'assets/teams/beatles.jpg'
    4;'AC/DC';'ac-dc';'Австралия';'1973';'хард-блюз-
    рок';'assets/teams/acdc.jpg'
    5;'Scorpions';'scorpions';'ФРГ';'1965';'хард-
    рок';'assets/teams/scorpions_.jpg'
    6;'Ленинград';'ленинград';'Россия';'1997';'ска, фолк,
    панк';'assets/teams/leningrad.jpg'`;
    // шаблон, разбивающий текст в массив построчно
    let reg = new RegExp("\n");
    // разбиваем строку по шаблону в массив
    let arr = text.split(reg);
    // вывод массива
    console.log(arr); //
    // вывод третьего элемента массива
    console.log(arr[2]); //
</script>
```

replace

```
str.replace(substr|regexp, newSubStr|function);
```

Метод **replace**() возвращает новую строку с некоторыми или всеми сопоставлениями **шаблона**, заменёнными на **заменитель**.

Шаблон может быть:

- **строкой**
- **регулярным выражением**

Заменитель может быть:

- **строкой**
- **функцией**, вызываемой при каждом сопоставлении.

> ***К сведению***. Возможности метода настолько велики, что рассмотреть их в одном или двух уроках практически невозможно. Продолжим изучение метода после темы пользовательских функций.

Параметры

- **substr** – строка, заменяемая на **newSubStr**. Обратите внимание, будет заменено только первое вхождение искомой строки.
- **regexp** – объект регулярного выражения **RegExp**. Сопоставление заменяется возвращаемым значением второго параметра.
- **newSubStr** – строка, заменяющая подстроку из первого параметра.
- **function** – функция, вызываемая для создания новой подстроки.

Возвращаемое значение

Новая строка с некоторыми или всеми сопоставлениями шаблона, заменёнными на заменитель.

example_11. Метод replace()

```
<script>
    // строка в которой будем искать совпадение с шаблоном
    let tags = document.querySelector("p").innerText;
    // шаблон поиска всех вхождений символа - "#"
    let reg = new RegExp("#", "g");
    // замена всех "#" на "<li>"
    let list = tags.replace(reg, "<li>");
```

```
    // формируем строку представляющую список
    list = `
        <h3>Изучаем Веб-технологии с pechora-PRO</h3>
        <ul>${list}</ul>
    `;
    // вывод списка в документ
    document.querySelector("div").innerHTML = list;
</script>
```

example_12. Метод replace()

```
<script>
    // строка в которой будем искать совпадение с шаблоном
    let text = document.querySelector("p").innerHTML;
    // глобальный шаблон поиска
    let reg = new RegExp("javascript", "g");
    // поиск и замена
    let edit_text = text.replace(reg, "<b>JavaScript</b>");
    // вставляем в документ измененный текст
    document.querySelector("div").innerHTML = edit_text;
    // вывод в консоль
    // console.log(edit_text);
</script>
```

Методы объекта RegExp

Все рассмотренные методы вызывались на **объекте String**(). Теперь рассмотрим методы, которые вызываются на **объекте RegExp**().

Еще раз. В отличие от предыдущих, следующие методы вызываются на объекте **регулярного выражения**.

test

```
reg.test(str);
```

Метод **test**() выполняет поиск сопоставления регулярного выражения **reg** указанной строке **str**.

Параметр

- **str** – строка, с которой сопоставляется регулярное выражение.

Возвращаемое значение

Возвращает логическое значение: **true** или **false**.

example_13. Метод test()

```
<script>
    // полный код смотреть в файле раздаточного материала
    // массив заказа
    let order = [ {/.../}, {/.../}, {/.../} ];
    // преобразуем массив заказов в строку
    let orderJSON = JSON.stringify(order);
    // вывод в консоль строки
    // console.log(orderJSON);
    // создание RegExp
    let reg = new RegExp("Парка", "g");
    // поиск в заказе товара - Парка
    let res = reg.test(orderJSON);
```

```
    // вывод результата поиска
    console.log(res);
</script>
```

exec

```
reg.exec(str);
```

Метод **exec**() ищет совпадение регулярного выражения **reg** в строке **str**.

Параметр

- **str** – строка, с которой производится сопоставление регулярного выражения.

Возвращаемое значение

Если сопоставление успешно выполнилось, метод **exec**() **возвращает массив**. Возвращаемый массив в **первом элементе** содержит сопоставленный текст, а в **последующих элементах** — текст, захваченный при сопоставлении круглыми скобками (рассмотрим в следующем уроке).

Если сопоставление не удалось, метод **exec**() **возвращает null**.

example_14. Метод exec()

```
<script>
    // строка, в которой будем искать совпадение с шаблоном
    let text = document.querySelector("pre h3").innerText;
    // шаблон поиска
    let reg = new RegExp("дале", "i");
    // возвращаемый массив
    let arr1 = reg.exec(text);
    // вывод массива arr1
```

```
    console.dir(arr1);
    // сравним поиск методом -match
    arr2 = text.match(reg);
    // вывод массива arr2
    console.dir(arr2);
    // массивы одинаковы
</script>
```

Метод **exec**() ведёт себя по-разному в зависимости от того, имеет ли регулярное выражение флаг **g**.

Если флага нет, то **exec**() возвращает первое совпадение в точности как **match**().

Если флаг **есть**, то:

- Вызов **exec**() возвращает **первое совпадение** и запоминает позицию после него в свойстве **lastIndex**.
- Следующий вызов начинает поиск с позиции **lastIndex**, возвращает следующее совпадение и запоминает позицию после него в **lastIndex**.
- …И так далее.
- Если совпадений больше нет, то **exec** возвращает **null**, а для **lastIndex** устанавливается значение 0.

Таким образом, повторные вызовы возвращают одно за другим все совпадения, используя свойство **lastIndex** для отслеживания текущей позиции поиска. Подробное описание такого поведения метода рассмотрено в примере **example_15**.

> ***Примечание***. **Многократный** вызов метода отлично реализуется с помощью цикла. Займемся этим в соответствующей теме.

example_15. Многократный вызов метода exec()

```
<script>
    // строка в которой будем искать совпадение с шаблоном
    let text = document.querySelector("pre h3").innerText;
    // шаблон поиска, используем флаг -g
    let reg = new RegExp("дале", "ig");
    // 1-ый поиск
    let arr = reg.exec(text);
    //
    console.log("\n\n1-ый поиск");
    console.log(arr);
    console.log("Сопоставление найдено в позиции ",
    arr["index"]);
    console.log("Следующий поиск с позиции ", reg.lastIndex);
    // 2-ой поиск
    arr = reg.exec(text);
    //
    console.log("\n\n2-ой поиск");
    console.log(arr);
    console.log("Сопоставление найдено в позиции ",
    arr["index"]);
    console.log("Следующий поиск с позиции ", reg.lastIndex);
    // 3-ий поиск
    arr = reg.exec(text);
    //
    console.log("\n\n3-ий поиск");
```

```
    console.log(arr); // null
    console.log("Сопоставление найдено в позиции ",
    arr["index"]);
    console.log("Следующий поиск с позиции ", reg.lastIndex);
</script>
```

Пример

Небольшой практический пример.

example_16. replace()

```
<script>
    // забираем из документа содержимое блока div
    var str = document.body.querySelector("div").innerHTML;
    // выполняем замену 1 вхождения
    var newstr = str.replace(/{{img1}}/, `<img src='pict1.jpg'
    width='250px'>`);
    // выполняем замену 2 вхождения
    newstr = newstr.replace(/{{img2}}/, `<img src='pict2.jpg'
    width='250px'>`);
    // выполняем замену 3 вхождения
    newstr = newstr.replace(/{{img3}}/, `<img src='pict3.jpg'
    width='250px'>`);
    // выполняем замену 4 вхождения
    newstr = newstr.replace(/{{img4}}/, `<img src='pict4.jpg'



    width='250px'>`);
```

```
    // выполняем замену 5 вхождения
    newstr = newstr.replace(/{{img5}}/, `<img src='pict5.jpg'
    width='250px'>`);
    // замещаем блок новым содержимым
    document.body.querySelector("div").innerHTML = newstr;
</script>
```

Похожие задачи выполняются не совсем так. Это можно сделать проще и элегантнее используя **метасимволы**.

Задание № 27

В директории раздаточного материала в файле **index.html**, вам предложен фрагмент **HTML-кода вывода изображений**.

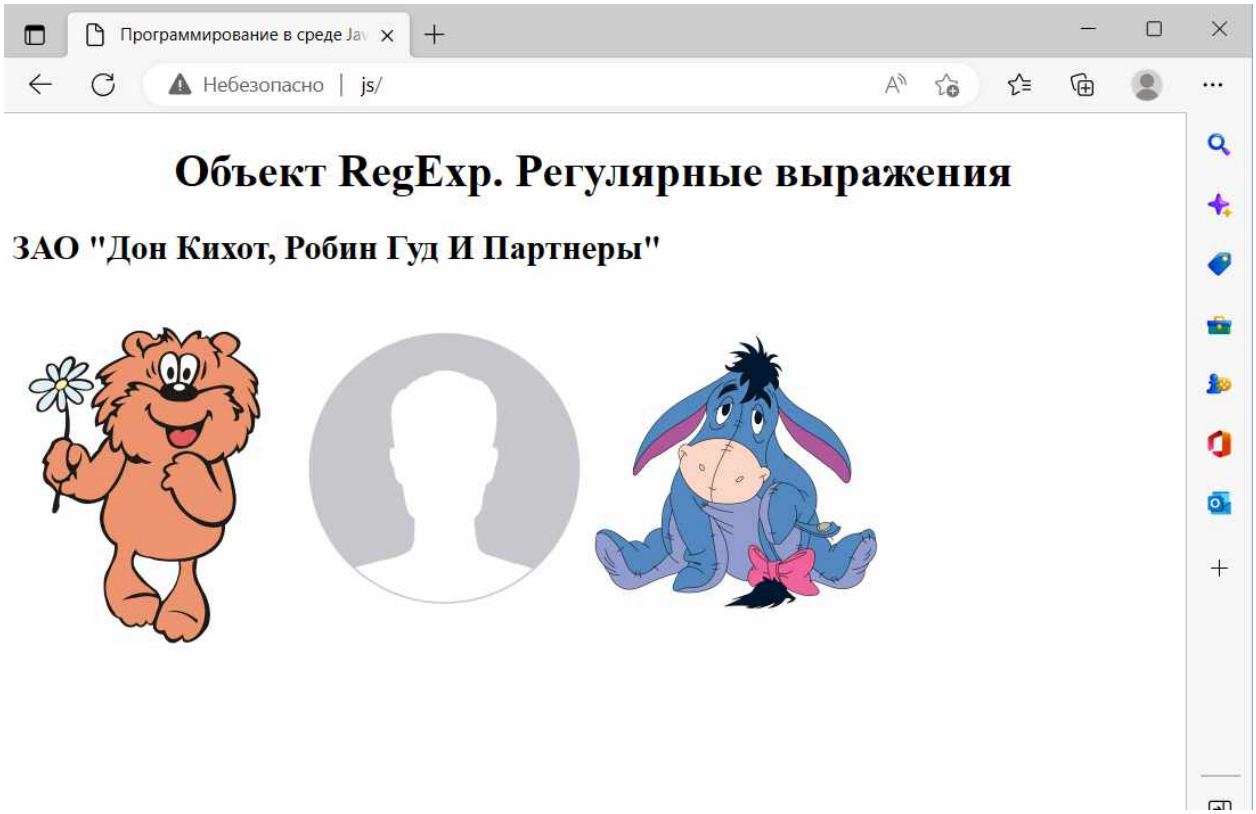
```
<div id="app">
    <img src='img/ID14772.jpg' width='200px'>
    <img src='img/ID23121.jpg' width='200px'>
    <img src='img/ID13425.jpg' width='200px'>
</div>
```

Используя возможность регулярных выражений, напишите скрипт, выполняющий замену одного из изображений на изображение, расположенное в **img/no-photo.jpg**.

Идентификатор **заменяемого** изображение указан в переменной скрипта.

```
<script>
    // идентификатор заменяемого изображения
    let id = "ID23121";
    // ...
</script>
```

Примерный формат вывода представлен на рисунке.


Программирование в среде Jav
Небезопасно | js/
Объект RegExp. Регулярные выражения
ЗАО "Дон Кихот, Робин Гуд И Партнеры"

Задание № 28

В раздаточной директории вам предложен файл **temp.js**, содержащий информацию о темпераментах. Информация структурирована в виде **массива объектов**, сохраненного в формате **JSON**.

```
// массив объектов темпераментов

let temp = `[

    {

    "title" : "Флегматик",

    "descr" : "Спокойный и медлительный, движения неторопливые, законченные, речь уверенная, аргументированная. Часто наблюдается массивное телосложение, широкая грудная клетка, крупные черты лица. Долго сходится с людьми, тяжело отпускает.",

    "img" : "{{img-phl}}"

    },

    { ... },

    { ... },

    { ... }

]`;
```

Подключите файл темпераментов к файлу скрипта **index.html**. Напишите скрипт, используя **возможности** регулярных выражений выполните следующие действия:

- замените строковые обозначения изображений на изображения из директории **pict**.
- распарсите **JSON** формат в **JS** объект
- выведите полученный контент в браузер.

Примерный формат представлен на рисунке. Можете использовать свой вариант.

Флегматик

Спокойный и медлительный, движения неторопливые, законченные, речь уверенная, аргументированная. Часто наблюдается массивное телосложение, широкая грудная клетка, крупные черты лица. Долго сходится с людьми, тяжело отпускает.

Сангвиник

Активный, общительный, открытый. Широкие короткие кости, устойчивая прямая осанка, круглая голова, внушительная по объему шея. Быстро знакомится и не менее стремительно разрывает отношения. Отличный организатор, лидер, амбициозный и любящий признание. Самый мощный стимул к работе — деньги.

Задание № 29

В раздаточной директории вам предложен файл **data.js**, содержащий информацию о некоторых музыкальных группах. Информация структурирована в виде строкового представления **CSV** формата. Изучите структуру данных.

Подключите файл к файлу скрипта **index.html**. Используя возможности **регулярных выражений** извлеките из CSV-строки информацию о группе **AC/DC**. Выведите полученную информацию в **браузер**.

Примерный формат вывода представлен на рисунке.