Работа со строками

Введение

В JavaScript любые **текстовые** данные являются **строками**. Несмотря на то, что это примитив, строка сложно устроена внутри.

Визуально строки состоят из символов. Символ, который видно на экране хранится в компьютере как одно или несколько чисел, каждое такое число называют **юнитом**. Компьютер хранит таблицу в которой **юниту** (числу) **соответствует символ**. Такие таблицы называют **кодировкой**.

Строка в JavaScript – это неизменяемая, упорядоченная последовательность 16-битных значений, каждое из которых представляет собой символ Unicode. Для представления символов Unicode в JavaScript используется кодировка UTF-16.

> ***Еще раз***. Внутренний формат для строк — всегда UTF-16, вне зависимости от кодировки страницы.

Каждый **символ в строке определяется по номеру индекса**, как и элементы массивов.

В JavaScript есть два вида строк:

- **примитивные**
- **объекты String**.

Как правило, на практике используется **примитивный** тип строк, так как JavaScript может использовать **свойства** и **методы** объекта String **без преобразования** примитивной строки в object-строку.

Объект String

Объект String используется, чтобы представить и конструировать последовательность символов.

String — это обёртка над **примитивным** строковым типом, которая содержит дополнительные методы работы со строками:

- **поиска** по строке;
- строковых **преобразований**;
- получения отдельных **символов**.

Строки **автоматически оборачиваются в обёртку String** при вызове методов над ними.

```
const primitive = 'привет, мир!';

// вызов метода на примитиве автоматически сделает из строки объект

console.log(primitive.toUpperCase()); // ПРИВЕТ, МИР!
```

Обернуть строку в String можно **вручную**, вызвав конструктор new String().

```
// примитивная строка

let primitive = 'Привет, Мир!';

// объект String

let str = new String('Привет, Мир!');
```

В этом случае переменные primitive и str будут **разных типов**.

Инициализируем оба типа строк и выведем их тип с помощью **typeof**.

example_1. Строки. Примитивы и объекты

```
<script>

     // инициализация примитивной строки

     const primitiveString = "Привет, Мир ;)";

     // инициализация String Object

     const objectString = new String("Привет, Мир ;)");

     // выведем типы строк в консоль

     console.log(typeof primitiveString); // string

     console.log(typeof objectString);  // object

</script>
```

> ***Еще раз***. Строки **автоматически оборачиваются в обёртку String** при вызове методов над ними. Это позволяет использовать **свойства** и **методы** объекта String без явного создания Object-строки (на примитиве).

Определение строк

В JavaScript любые текстовые данные являются **строками**. **Строка** — это ноль или больше символов, заключенных в кавычки. Строки используются для хранения и манипулирования текстовыми данными.

Есть несколько способов создать строку:

- **одинарными** кавычками

- **двойными** кавычками
- **обратными** кавычками (шаблонная строка).

Пример:

```
let single = 'single-quoted';

let double = "double-quoted";

let backticks = `backticks`;
```

Записи в одинарных и двойных кавычках, по сути, идентичны. Шаблонные строки позволяют подставлять в строку **значения переменных**. В местах, где нужно вставить значение из переменной используется синтаксис **${имя_переменной}**.

Для лучшей читаемости программисты, как правило, редко пишут строчки кода длиннее 80 символов (рекомендация разработчиков). Если строковое выражение JavaScript не помещается на одной строке, то его можно перенести на новую строку. Ещё одно преимущество обратных кавычек — они могут **занимать более одной строки**.

example_2. Обратные кавычки и интерполяция переменных

```
<script>

    // выберем целевой элемент

    let h2 = document.querySelector("h2");

    // инициируем переменные

    let interpreter = "В. Санович";

    let scheme = "6-7-7-9-9";

    let genre = "Танка";

    let collection = "Хякунин иссю (Сто стихотворений ста
    поэтов)";

    // инициируем строку,

    // выполняем вставку переменных
```

```
    let txt = `
        <b><p>Я жалею людей <br />
        Я жалею людей. <br />
        Я презираю людей. <br />
        Я отчаялся думать <br />
        О печалях этого мира <br />
        И в свою печаль погрузился. </b><p>
        <hr>
        Переводчик: <i>${interpreter}</i> <br />
        Схема: <i>${scheme}</i> <br />
        Жанр: <i>${genre}</i> <br />
        Изборник: <i>${collection}</i>
    `;
    // вставляем фрагмент в документ
    h2.insertAdjacentHTML('afterend', txt);
</script>
```

При попытке использовать в многострочных строках **одинарные** или **двойные** кавычки наподобие обратных – появится ошибка:

```
// Uncaught SyntaxError: Invalid or unexpected token
let txt = "
    Сколько той жизни?
    А всё не могу пройти
    Мимо сирени..
";
```

Если все-таки есть необходимость использовать многострочный текст с одинарными / двойными кавычками, можно сделать следующим образом:

```
document.getElementById("demo").innerHTML =
"Яркий лунный свет!<br/>На циновку тень свою<br/>Бросила сосна."
```

Также, можно разбить строку при помощи символа обратного слеша (\):

```
let demo = document.getElementById("demo");
demo.innerHTML = "Ночью под снегом <br/>\
Спят, прижавшись друг к другу,<br/>\
Горы Синано.";
```

Однако это не лучший способ, так как у него нет универсальной поддержки. Некоторые браузеры не допускают использование пробелов после символа \.

Наиболее безопасным способом переноса строк является оператор **строкового сложения** (**конкатенация** строк):

```
let demo = document.getElementById("demo");
demo.innerHTML = "Сегодня утром <br/>" +
"Тихонько упал на землю <br/>" +
"С дерева лист.";
```

Одинарные и двойные кавычки в языке используются очень давно, тогда потребность в многострочных строках не учитывалась. Обратные кавычки появились существенно позже, и поэтому они гибче.

Внутри строки также **можно использовать кавычки**, если они отличаются от кавычек, в которые заключена строка:

example_3. Использование кавычек внутри строк

```
<script>
    // выберем целевой элемент
    let h2 = document.querySelector("h2");
    let txt1 = "<h3>Однажды Шарлотта Вейл сказала: Oh, Jerry,
    don't let's ask for the moon. We have the stars</h3>";
    let txt2 = '<h3>Однажды Дон Корлеоне сказал: I"m gonna make
```

```
    him an offer he can"t refuse</h3>';
    let txt3 = `<h3>Однажды Терминатор сказал: 'I'll be
    back'</h3>`;
    // вставляем фрагмент в документ
    h2.insertAdjacentHTML('afterend', txt1);
    h2.insertAdjacentHTML('afterend', txt2);
    h2.insertAdjacentHTML('afterend', txt3);
</script>
```

Специальные символы

Кавычки, используемые **внутри** строки, должны отличаться от кавычек, **определяющих** строку. JavaScript генерирует ошибку при чтении следующих строк:

```
str1 = "Oh, Jerry, don"t let"s ask for the moon. We have the stars";
str2 = 'I'm gonna make him an offer he can't refuse';
```

Для решения этой проблемы можно воспользоваться **экранирующим символом**. Экранирующий символ (\) **преобразует специальные символы в символы строки**:

Символ	Результат	Описание
'	'	Одинарная кавычка
\"	"	Двойная кавычка
****	\	Обратный слэш
\n		Перенос строки
\t		Знак табуляции

Теперь все работает:

```
str1 = "Oh, Jerry, don\"t let\"s ask for the moon. We have the stars";

str2 = 'I\'m gonna make him an offer he can\'t refuse';

console.log(str1);

console.log(str2);
```

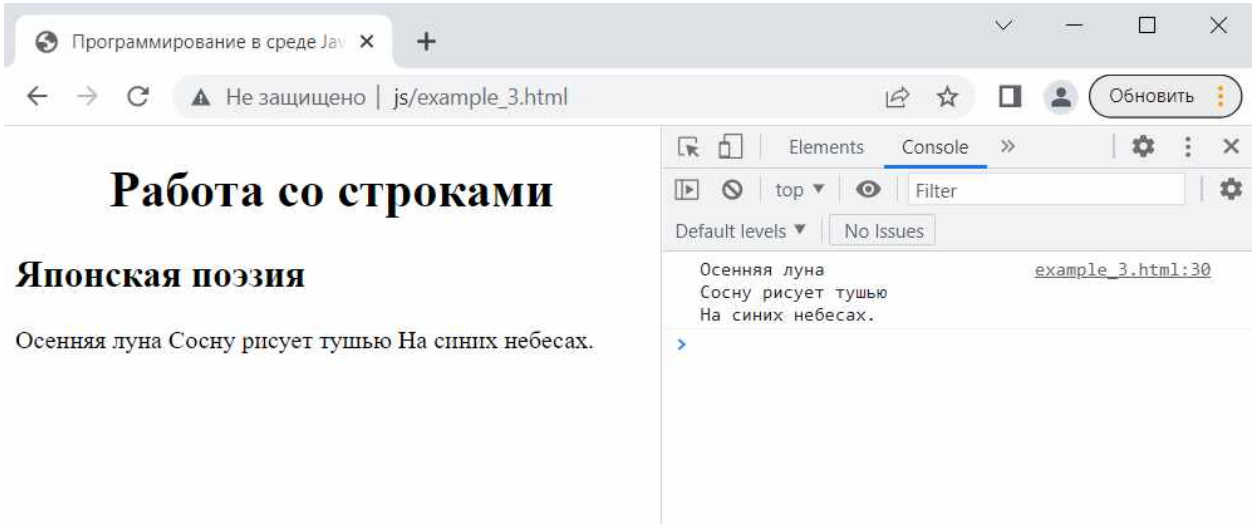
Многострочные строки также можно создавать с помощью одинарных или двойных кавычек, используя так называемый **символ переноса строки**, который записывается как **\n**.

```
// в качестве разделителя используем символ перевода строки

let hokku = "Осенняя луна\nСосну рисует тушью\nНа синих небесах.";
```

При использовании символа **переноса строки** вы должны четко представлять результат вставки строки в браузер. Символ \n это не
, пример смотрите в example_4.

example_4. Применение символа переноса строк

```
<script>

    let pre = document.getElementsByTagName("p")[0];

    // формируем строку,

    // в качестве разделителя - символ переноса строки

    let hokku = "Осенняя луна\nСосну рисует тушью\nНа синих небесах."

    // браузер переносов не видит, ему нужны теги

    pre.innerHTML = hokku;

    // консоль понятливее ;)

    console.log (hokku);

</script>
```



При разной форме определения, формируемые строки остаются **эквивалентными**. Внимательно проанализируйте следующий пример, не забывайте, пробел такой же символ, как любой другой.

example_5. Пример эквивалентности строк

```
<script>
    // перевод строки добавлен с помощью символа перевода строки
    let hokku1 = "Снова весна.\nПриходит новая глупость\nСтарой
    на смену.";
    // многострочная строка, созданная с использованием обратных
    кавычек
    let hokku2 = `Снова весна.
    Приходит новая глупость
    Старой на смену.`;
    // сложение строк
    let hokku3 = "Снова весна.\n" +
    "Приходит новая глупость\n" +
    "Старой на смену.";
    // визуальное отображение строк
    console.log (hokku1);
```

```
    console.log (hokku2);
    console.log (hokku3);
    // сравнение строк
    console.log('hokku1 === hokku2', hokku1 === hokku2); // true
    console.log('hokku1 === hokku3', hokku1 === hokku3); // true
    console.log('hokku2 === hokku3', hokku2 === hokku3); // true
</script>
```

Напомню, что при определении двух переменных:

```
str1 = "100";
str2 = 100;
// простое сравнение даст true
console.log(str1 == str2); // true
// тождественное сравнение даст false
console.log(str1 === str2); // false
```

Поиск подстроки

Существует несколько способов поиска подстроки:

- **indexOf**
- **lastIndexOf**
- **search**
- **includes**
- **startsWith**, **endsWith**

indexOf

```
str.indexOf(substr[, pos]);
```

Метод **indexOf**() ищет подстроку **substr** в строке **str**, начиная с позиции **pos**, и возвращает позицию (индекс), на которой располагается совпадение, либо -1 при отсутствии совпадений.

Параметры

- **substr** – строка, представляющая искомое значение.
- **pos** – необязательный параметр. Местоположение внутри строки, откуда начинать поиск. Может быть любым целым числом.

Метод indexOf() является **регистрозависимым**.

```
var str = "Welcome pechora-PROger!";
console.log(str.indexOf("pechora")); // 8
// отсчет с начала строки,
// но поиск фактически в строке "hora-PROger!"
console.log(str.indexOf("pechora", 10)); // -1
```

example_6. Метод indexOf

```
<script>
    let str =
    "Англичанин мне друг" +
    "Но пришлось утопить..." +
    "Пакетик чая"
    // проверяем вхождение подстрок
    console.log (str.indexOf("друг")); // 15
    console.log (str.indexOf("враг")); // -1
    console.log (str.indexOf("хуже татарина")); // -1
</script>
```

lastIndexOf

```
str.lastIndexOf(substr[, pos]);
```

Метод **lastIndexOf**() возвращает позицию (индекс) последнего вхождения заданного текста в строке, или -1, если ничего не было найдено. Поиск по строке ведётся от конца к началу, начиная с индекса **pos**.

Параметры

- **substr** – трока, представляющая искомое значение.
- **pos** – необязательный параметр. Местоположение внутри строки, откуда начинать поиск, **нумерация индексов идёт слева направо**. Может быть любым целым числом.

Метод lastIndexOf() является **регистрозависимым**.

```
var str = "Welcome pechora-PROger!";
console.log(str.lastIndexOf("pechora")); // 8
// поиск фактически в строке "Welcome pec"
console.log(str.lastIndexOf("pechora", 10)); // 8
console.log(str.lastIndexOf("pec", 10)); // 8
```

Внимательно посмотрите следующий пример, при внешней простоте методов, есть нюансы.

example_7. Метод lastIndexOf

```
<script>
    // поиск индекса заглавной 'O', ближайшей к концу строки
    // такая буква в слове - 'PROger'
    // отсчет начинается слева, с позиции - 0
    console.log("Welcome pechora-PROger".lastIndexOf("O"));
    // 18
    // поиск индекса строчной 'o', ближайшей к концу строки
    // такая буква в слове - 'pechora'
```

```
    // отсчет начинается слева, с позиции - 0
    console.log("Welcome pechora-PROger".lastIndexOf("o"));
    // 12
    // поиск индекса строчной 'o', ближайшей к концу строки
    // причем, строка обрезается вторым аргументом
    // фактически поиск идет в строке 'Welcome pec'
    // тогда, ближайшая к концу строки 'o' в слове 'Welcome'
    console.log("Welcome pechora-PROger".lastIndexOf("o", 10));
    // 4
</script>
```

Оба метода **indexOf**() и **lastIndexOf**() возвращают -1, если текст не найден. Оба метода могут принимать **второй параметр** в качестве начальной позиции для поиска.

В качестве проверки попробуйте определить индексы вхождения следующего примера. В случае затруднения смотрите самостоятельную работу 3_6.

example_8. Второй параметр в методах indexOf и lasIndexOf

```
<script>
    // получаем строку из документа
    var str = document.querySelector("pre h3").innerText;
    console.log(str.indexOf("дале")); // 67
    console.log(str.lastIndexOf("дале")); // 85
    // определить индексы до запуска скрипта
    // console.log(str.indexOf("дале", 70)); // ??
    // console.log(str.lastIndexOf("дале", 70)); // ??
</script>
```

search

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

Метод **search**() возвращает индекс первого сопоставления с регулярным выражением внутри строки. В противном случае метод вернёт -1.

Параметр

- **regexp –** объект регулярного выражения. Если будет передан не объект регулярного выражения, а простая строка, то она будет неявно преобразована.

> ***Примечание***. Работа с **регулярными выражениями** чрезвычайно важна не только в JavaScript, но в любом языке программирования и рассматривается в отдельном уроке.

```
var str = "Welcome pechora-PROger!";
console.log(str.search("pechora")); // 8
```

example_9. Метод search

```
<script>
     // определяем строку
     str = `И той порою,
          Когда непринуждённо
          Дождь смывал следы,
          Убогими казались мысли
          О проходящем...`;
     console.log(str.search("дождь")); // -1
     // 'Дождь' <> 'дождь'
</script>
```

Различия методов search() и indexOf():

- метод **search**() не может принимать второй аргумент начального положения.
- метод **indexOf**() не может принимать в качестве аргумента регулярные выражения.

В настоящее время метод **indexOf**() считается устаревшим и может встретиться только в старом коде, в новом он просто не нужен: ему на смену пришел более современный метод **includes**().

includes

```
str.includes(substr[, pos]);
```

Метод **includes**() возвращает **true**, если в строке **str** есть подстрока **substr**, либо **false**, если нет.

Параметры

- **substr** – подстрока для поиска в данной строке.
- **pos** – необязательный параметр. Позиция в строке, с которой начинать поиск строки **substr**, по умолчанию 0.

Метод используется, если необходимо проверить, есть ли совпадение, но **позиция не нужна**. Необязательный второй аргумент **includes**() позволяет начать поиск с определённой позиции.

```
var str = "Welcome pechora-PROger!";
console.log(str.includes("pechora")); // true
```

example_10. Метод includes

```
<script>
    // определяем строку
    str = document.getElementsByTagName("h3")[0].innerText;
    // ищем вхождение
    console.log(str.includes("сережки")); // true
```

```
    // а так?
    console.log(str.includes("серёжки")); // ???
</script>
```

Разница между методами **indexOf** и **includes**:

```
var str = "pechora-PRO лучший ресурс начинающего разработчика";
// в условных конструкциях 0 фактически это false
console.log(str.indexOf("pechora")); // 0
console.log(str.includes("pechora")); // true
```

startsWith, endsWith

```
str.startsWith(substr[, pos]);
```

Метод **startsWith**() проверяет, начинается ли строка с символов указанных в скобках, возвращая, соответственно, **true** или **false**.

Параметры

- **substr** – символы, искомые в начале данной строки.
- **pos** – необязательный параметр. Позиция в строке, с которой начинать поиск substr, по умолчанию 0.

```
str.endsWith(substr[, length]);
```

Метод **endsWith**() проверяет, заканчивается ли строка символами указанными в скобках, возвращая, соответственно, **true** или **false**.

Параметры

- **substr** – символы, искомые в конце строки.
- **length** – необязательный параметр. Позволяет искать внутри строки, обрезая её по диапазону, по умолчанию равен длине строки.

example_11. Методы startsWith, endsWith

```
<script>

     // определяем строку

     var str = "Welcome to pechora-proger group";

     // тестируем методы

     console.log(str.startsWith("Welcome")); // true

     console.log(str.startsWith("welcome")); // false

     console.log(str.endsWith("proger")); // false

     console.log(str.endsWith("proger", 25)); // true

</script>
```

Извлечение части строки

Существует три способа извлечения части строки:

- **slice**
- **substring**
- **substr**

slice

```
str.slice(start[, end]);
```

Метод **slice**() возвращает часть строки от **start** до (не включая) **end**. Метод принимает два параметра.

Параметры

- **start** – индекс, с которого начинать извлечение (нумерация начинается с нуля)
- **end** – индекс, **перед** которым заканчивать извлечение (нумерация начинается с нуля).

```
var str = "Welcome to pechora-proger group";

console.log(str.slice(11, 25)); // pechora-proger
```

Если параметр имеет отрицательное значение, позиция учитывается от конца строки.

```
var str = "Welcome to pechora-proger group";

console.log(str.slice(-20, -13)); // pechora
```

Если опустить второй параметр, метод выполнит срез оставшейся части строки:

```
var str = "Welcome to pechora-proger group";

console.log(str.slice(11)); // pechora-proger group

console.log(str.slice(-5)); // group
```

substring

```
str.substring(start [, end]);
```

Метод **substring**() аналогичен методу **slice**(). Возвращает подстроку строки между двумя индексами, или от одного индекса и до конца строки.

Разница заключается в том, что **substring**() **не может принимать отрицательные индексы**. Если опустить второй параметр, substring() будет разрезать оставшуюся часть строки.

```
var str = "Welcome to pechora-proger group";

console.log(str.substring(0, 6)); // Welcome

console.log(str.substring(11)); // pechora-proger group

console.log(str.substring(-5)); // Welcome to pechora-proger group
```

example_12. Методы slice, substring

```
<script>

    // полный сотовый номер
```

```
    let phone = "8(912)552-92-25";
    // страна
    let country = phone.slice(0,1);
    // оператор
    let operator = phone.slice(2,4);
    // регион (может не соответствовать реальному)
    let region = phone.substring(4,5);
    // абонент
    let abonent = phone.substring(6);
    // формируем строку для вывода
    let out = `
        <h3>Информация о владельце номера</h3>
        Номер: <b>${phone}</b> <br/>
        Международный код страны: ${country} <br/>
        Оператор сотовой связи: ${operator} <br/>
        Регион: ${region} <br/>
        Абонент: ${abonent}
    `;
    // получаем доступ к элементу h2
    var el = document.querySelector("h2");
    // вставляем строку в документ
    el.insertAdjacentHTML("afterend", out);
</script>
```

substr

```
str.substr(start [, length]);
```

Метод **substr**() похож на **slice**(). Разница в том, что второй параметр указывает **длину** извлеченной подстроки.

Параметры

- **start** – позиция, с которой начинать извлекать символы.
- **length** – необязательный параметр. Количество извлекаемых символов.

```
var str = "Welcome to pechora-proger group";

console.log(str.substr(11, 7)); // pechora
```

Если опустить второй параметр, **substr**() будет извлекать **оставшуюся часть** строки.

```
var str = "Welcome to pechora-proger group";

console.log(str.substr(11)); // pechora-proger group
```

Если первый параметр отрицательный, позиция рассчитывается от конца строки.

```
var str = "Welcome to pechora-proger group";

console.log(str.substr(-12)); // proger group
```

example_13. Метод substr

```
<script>

    // статьи для вывода, в виде массива объектов

    // могли быть получены, например, от сервера при асинхронном
    запросе

    // каждый объект - фрагмент информации о статье

    let article = [

        {

            teazer: "...",

            href: "..."

        },
```

```
        {
            teazer: "...",
            href: "..."
        },
        {
            teazer: "...",
            href: "..."
        }
];
// формируем из массива строки-ссылки для вывода
let p1 = `<a href='${article[0].href}'
target='blank'>${article[0].teazer.substr(0,31)} ...</a>`;
let p2 = `<a href='${article[1].href}'
target='blank'>${article[1].teazer.substr(0,31)} ...</a>`;
let p3 = `<a href='${article[2].href}'
target='blank'>${article[2].teazer.substr(0,31)} ...</a>`;
// формируем список из ссылок
let out = `
    <ul>
    <li>${p1}</li>
    <li>${p2}</li>
    <li>${p3}</li>
    </ul>
`;
// вставляем список в документ
document.body.querySelector("h2").insertAdjacentHTML("aftere
nd", out);
```

```
</script>
```

Замена содержимого строки

replace

```
str.replace(regexp|substr, newsubstr[, flags]);
```

Метод **replace**() заменяет указанное значение другим значением в строке. Метод replace() **не изменяет строку**, в которой он вызывается, а возвращает **новую строку**.

Параметры

- **regexp** | **substr** – объект регулярного выражения RegExp или строка, заменяемая на **newsubstr**. Обратите внимание, будет заменено только **первое вхождение** искомой строки.
- **newsubstr** – строка, заменяющая подстроку из первого параметра.
- **flags** – необязательный параметр. Строка, задающая комбинацию флагов регулярного выражения.

```
var str = "[...] лучший ресурс для начинающего разработчика!";

str.replace("[...]", "pechora-PRO");

console.log(str);
```

По умолчанию функция **replace**() заменяет только первое совпадение.

По умолчанию метод **replace**() **регистрозависим**.

example_14. Метод replace

```
<script>

    // забираем из документа содержимое блока div

    var str = document.body.querySelector("div").innerHTML;

    // выполняем замену
```

```
    var newstr = str.replace(/{{img}}/, "<img src='img.jpg'
    width='350px'>");
    // это самое простое применение метода .replace
    // возможности метода безграничны (почти;)
    // замещаем блок новым содержимым
    document.body.querySelector("div").innerHTML = newstr;
</script>
```

К сведению. Метод **replace** имеет практически неограниченные возможности по **манипулированию строками**. Рассмотрим метод более подробно в уроке, посвященном регулярным выражениям.

Преобразование строк

Преобразование в верхний и нижний регистр

toUpperCase

```
str.toUpperCase();
```

Метод **toUpperCase**() возвращает значение строки, преобразованное в верхний регистр. Метод toUpperCase() не изменяет значение самой строки.

```
var str = "Welcome pechora-PROf!";
var upstr = str.toUpperCase();
alert(upstr); // WELCOME PECHORA-PROF!
alert(str); // Welcome pechora-PROf!
```

toLowerCase

```
str.toLowerCase();
```

Метод **toLowerCase**() возвращает значение строки, преобразованное в нижний регистр. Метод toLowerCase() не изменяет значение самой строки.

```
var str = "Welcome pechora-PROf!";

var lowstr = str.toLowerCase();

alert(lowstr); // welcome pechora-prof!

alert(str); // Welcome pechora-PROf!
```

Преобразование в массив

split

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

Метод **split**() разбивает объект String на массив строк путём разделения строки указанной подстрокой.

Параметры

- **separator** – необязательный параметр. Указывает символы, используемые в качестве разделителя внутри строки.
- **limit** – необязательный параметр. Целое число, определяющее ограничение на количество найденных подстрок.

Метод split() возвращает новый массив.

```
var str = "Ave, Caesar, morituri te salutant";

str.split(","); // ['Ave', ' Caesar', ' morituri te salutant']

str.split(" "); // ['Ave,', 'Caesar,', 'morituri', 'te',
'salutant']
```

Если разделитель опущен, возвращаемый массив будет содержать всю строку в **позиции [0]**.

Если разделитель **""**, возвращаемый массив будет массивом, в котором, каждый элемент состоит из **одного символа**.

example_15. Разбиваем строку в массив

```
<script>
	// забираем содержимое тега h2,
	// вместе со всеми переводами строк и табуляциями
	let str = document.querySelector("pre h2").innerText;
	// разделитель - перенос строки: \n
	let arr = str.split("\n");
	// вывод массива
	console.log(arr);
	// разделитель - запятая: ,
	arr = str.split(",");
	// вывод массива
	console.log(arr);
	// разделитель - знак табуляции: \t
	arr = str.split("\t");
	// вывод массива
	console.log(arr);
</script>
```

Некоторые полезные возможности

Извлечение строковых символов

Существует два безопасных метода для извлечения строковых символов:

- **charAt**
- **charCodeAt**

charAt

```
str.charArt(index);
```

Метод **charAt**() возвращает символ по указанному **индексу** (позиции) в строке.

Параметр

- **index** – Целое число от 0 до длины строки минус 1.

Символы в строке идут слева направо. Индекс первого символа равен 0, а последнего символа в строке stringName равен **stringName.length - 1**. Если предоставленный вами параметр **index** выходит за пределы этого диапазона, JavaScript вернёт **пустую строку**.

```
var str = "Hello, User!";

var ch = str.charAt(0);

alert(ch); // H
```

charCodeAt

```
str.charCodeArt(index);
```

Метод **charCodeAt**() возвращает **код символа** по указанному индексу в строке.

Параметр

- **index** – целое число больше, либо равное 0 и меньше длины строки; если параметр не является числом, он устанавливается в 0.

Метод charCodeAt() возвращает **NaN**, если указанный индекс меньше нуля или больше длины строки.

```
var str = "Hello, User!";

var code = str.charCodeAt(0);

alert(code); // 72
```

trim

```
str.trim();
```

Метод **trim**() возвращает строку с вырезанными пробельными символами с её концов. Метод trim() не изменяет значение самой строки.

Пробельными символами считаются все собственно пробельные символы (пробел, табуляция, неразрывный пробел и прочие).

```
var str = "       Hello, User!        ";

alert(str.trim());  // "Hello, User!"
```

length

```
str.length;
```

Свойство **length** возвращает длину строки.

```
var str = "ABCDEFGH";

alert(str.length);  // 8
```

example_16.

```
<script>

    // определение строк

    var en_message = 'pechora-PROf';

    var ru_message = "Привет, мы из Печоры!";

    var empty = '';

    // вывод строк в браузер

    document.write('Строка "' + en_message + '" имеет длину ' +
```

```
    en_message.length + ' символов <p>');

    document.write('Строка "' + ru_message + '" имеет длину ' +
    ru_message.length + ' символ <p>');

    document.write('Пустая строка имеет длину, равную ' +
    empty.length);

</script>
```

Задание № 6

Изучите **представленный код** скрипта файла раздаточного материала **index.html**. Проанализируйте работу методов **indexOf** и **lastIndexOf**. Убедитесь в правильности расчетов индексов приведенного примера.

Составьте **свой вариант** строковой переменной для демонстрации работы методов.

Задание № 7

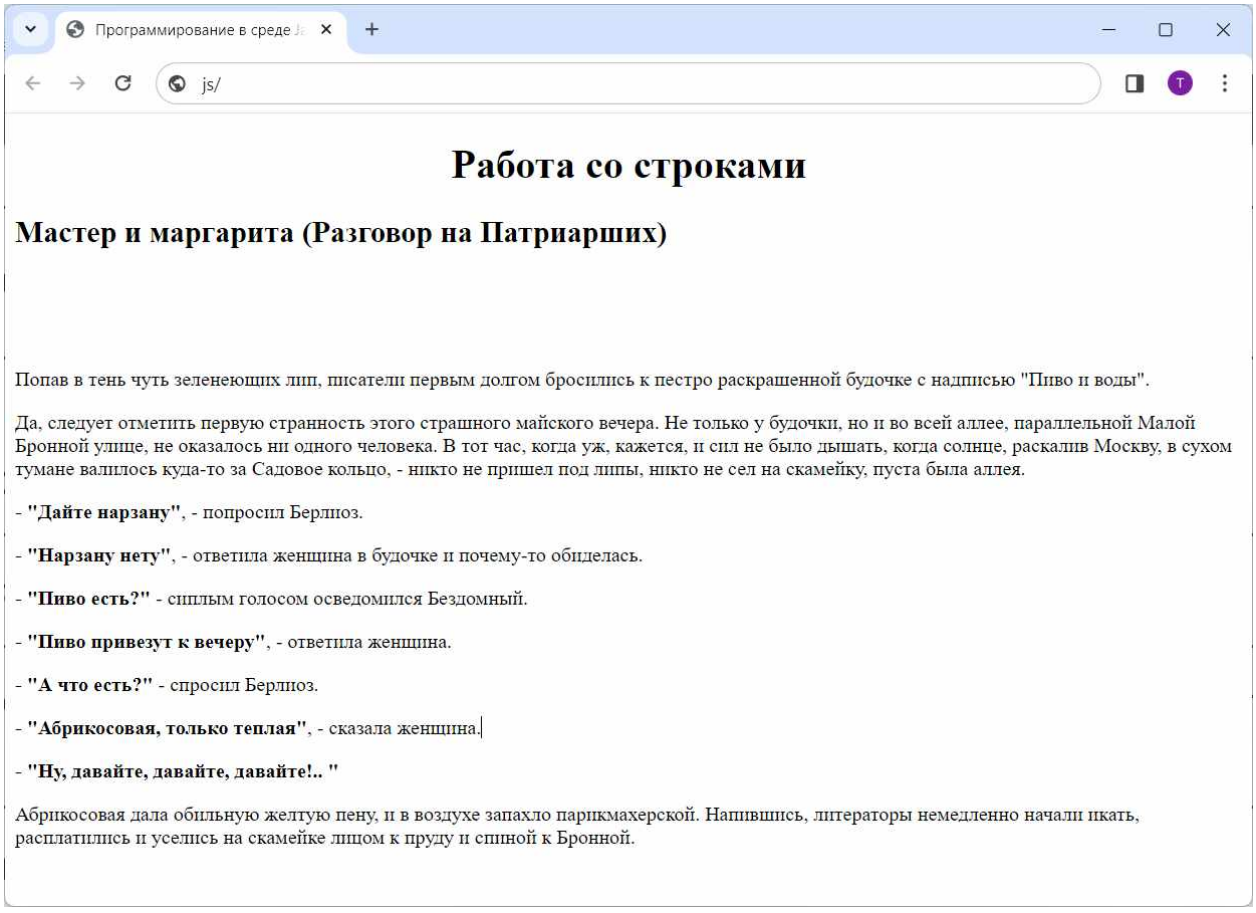
В директории раздаточного материала вам предложен документ **диалог**
В документе находится фрагмент из произведения М.А. Булгакова "Мастер и Маргарита", состоящий из **десяти** (10) абзацев.

Напишите скрипт, используя **конкатенацию строк** в **двойных кавычках** запишите все абзацы **в одну переменную**.

```
let str = "абзац 1" +

"абзац 2" +

"абзац 3" + ...
```

Используя метод **document.write**, **выведите** переменную в браузер.

Примерный **формат** вывода представлен на рисунке.



Задание № 8

В директории раздаточного материала в файле **index.html** предложен **фрагмент кода,** сравнивающий строки. При внешней схожести определения, формируемые строки **не эквивалентны**.

Проанализируйте код, исправьте "ошибку" разработчика – добейтесь **эквивалентности** всех сравниваемых строк.

Задание № 9

В директории раздаточного материала находится **скрипт вывода** условного заказа.

```
// формируем строку для вывода
// в строку подставляем значения массива
let str = `
<span>Идентификатор товара:</span> <b>${order[0]}</b><br/>
    <span>Название:</span> <b>${order[1]}</b><br/>
    <span>Бренд:</span> <b>${order[2]}</b><br/>
    <span>Цвет:</span> <b>${order[3]}</b><br/>
    <span>Страна:</span> <b>${order[4]}</b><br/>
    <span>Капюшон:</span> <b>${order[5]}</b><br/>
    <span>Размер:</span> <b>${order[6]}</b><br/>
    <span>Цена:</span> <b>${order[7]}</b><p>
    <img src="${order[8]}" width="250">
`;
```

Перепишите **фрагмент кода** формирования строки вывода.

Пусть строка для вывода формируется с помощью **конкатенации** строк, оформленных либо в **двойные,** либо в **одинарные** кавычки.

Задание № 10

В директории раздаточного материала находится файл **teams.js**, содержащий строковую переменную, хранящую данные в классическом **CSV-формате**.

> ***Примечание***. **CSV-файлы** (файлы данных с разделителями-запятыми) — представляют собой простые текстовые файлы, могут содержать только цифры и буквы и структурировать данные, содержащиеся в них, в табличной форме.
> Текст и числа, сохраненные в CSV-файле, можно легко переносить из одной программы в другую.

Напишите скрипт получения и вывода в консоль данных о группе **Scorpions**.

Править подключаемый файл **teams.js запрещено** (предполагаем, что именно в таком формате получили данные от сервера).

Примерный формат вывода отображен на рисунке.