|  |  |
| --- | --- |
| **1.Конкатенация строк**  В JavaScript конкатенация (объедение) двух или более строк в одну выполняется с помощью оператора +:  const firstName = 'Marcus';  const lastName = 'Wolfe';  **const fullName = firstName + ' ' + lastName;**  console.log(fullName); // "Marcus Wolfe"  Другой способ сложения строк в JavaScript — это использовать метод concat():  **const fullName = firstName.concat(' ', lastName);**  или (шаблонные строки для конкатенации строк)  **const fullName = `${firstName} ${lastName}`;**  Шаблонные строки — это способ создания строк с помощью обратных кавычек (`). Они появились в языке существенно позже, чем обычные строки и имеют в отличие от них ряд дополнительных возможностей.  **2.Длина строки**  Получение длины строки, т.е. количества символов из которых она состоит, выполняется в JavaScript с помощью свойства length:  const mystr = 'Моя строка текста';  // получаем количество символов  const len = mystr.length;  console.log(len); // 17  **3. Доступ к символам**  Извлечь определённый символ строки можно с помощью квадратных скобок. Внутрь квадратных скобок необходимо поместить число (порядковый номер символа). Отсчет символов в строке начинается с 0:  const str = 'Welcome';  // получим первый символ  console.log(str[0]); // "W"  // получим последний символ  console.log(str[str.length - 1]); // "e"  При передаче индекса несуществующего символа, получим undefined  **4.Методы для поиска подстроки в строке:**  indexOf(substring [, start]) – возвращает индекс, с которого начинается искомая подстрока в строке;  lastIndexOf(substring, [, start]) – то же самое, что indexOf, только поиск ведётся с конца строки.  const vegetables = 'Tomato, Potato, Carrot, Tomato';  vegetables.indexOf('Tomato', 5); // 24 (поиск (подстроки) Tomato начинается в 5 символа.  **5. includes:**  const vegetables = 'Tomato, Potato, Carrot, Tomato';  vegetables.includes('Carrot'); // true - Этот метод возвращает true, если искомая строка найдена. В противном случае, false.  **6. Методы toLowerCase и toUpperCase** предназначены для преобразования соответственно символов строки в строчные и прописные.  let myString = "Welcome to itchief!";  let upperCaseString = myString.toUpperCase();  let lowerCaseString = myString.toLowerCase();  console.log(upperCaseString); // "WELCOME TO ITCHIEF!"  console.log(lowerCaseString); // "welcome to itchief!" | **7. Методы для получения подстроки:**  **substring(start [, end])** – часть строки между start и end (не включая);  const greeting = 'Hello, Bob! How are you?';  // если второй аргумент не указан, то до конца строки  greeting.substring(7); // "Bob! How are you?"  greeting.substring(7, 10); // "Bob"  **substr(start [, length])** – часть строки, которая начинается с индекса start длиной length;  const greeting = 'Hello, Bob! How are you?';  // если второй аргумент не указан, то до конца строки  greeting.substr(7); // "Bob! How are you?"  // второй аргумент отвечает за количество возвращаемых символов  greeting.substr(7, 3); // "Bob"  **slice(start [, end])** – часть строки между start и end (не включая);  // если второй аргумент не указан, то до конца строки  greeting.slice(7); // "Bob! How are you?"  // до end (не включая)  greeting.slice(7, 10); // "Bob"  **8. replace**(pattern, replacement);  **replaceAll**(pattern, replacement).  В качестве pattern можно использовать строку или регулярное выражение, а для replacement – строку или функцию.  const nums = 'One, Two, Three, Two';  // заменим первую найденную строку "Two" на "Six"  nums.replace('Two', 'Six'); // One, Six, Three, Two  // заменим все найденные строки "Two" на "Six"  nums.replaceAll('Two', 'Six'); // One, Six, Three, Six  В этом примере с помощью метода replace заменили в nums подстроку 'Two' на 'Six'. Для этого мы в качестве первого аргумента передали в replace строку, которую нужно заменить. А во второй аргумент строку, на которую её нужно заменить. С помощью replaceAll мы заменили все вхождения 'Two' на'Six'.  Использование **replaceAll** для удаления пробелов из строки:  const str = 'A B C';  **const strWithoutSpaces = str.replaceAll(' ', '');**  console.log(strWithoutSpaces); // ABC  Если нужно убрать другой символ из строки:  **'Football'.replaceAll('o', ''); // "Ftball"**  **9. Метод repeat** позволяет создать строку, состояющую из указанного количества повторений другой:  'Bob! ' + 'Yes! '.repeat(3); // "Bob! Yes! Yes! Yes!"  10. Удаление пробелов trim()  Str.trim()  **11. startsWith и endsWith:**  const href = 'https://itchief.ru/javascript/';  href.startsWith('https://itchief.ru'); // true  const file = 'index.html';  file.endsWith('.html'); // true |

**1.Конкатенация строк**

В JavaScript конкатенация (объедение) двух или более строк в одну выполняется с помощью оператора +:

const firstName = 'Marcus';

const lastName = 'Wolfe';

**const fullName = firstName + ' ' + lastName;**

console.log(fullName); // "Marcus Wolfe"

Другой способ сложения строк в JavaScript — это использовать метод concat():

**const fullName = firstName.concat(' ', lastName);**

или (шаблонные строки для конкатенации строк)

**const fullName = `${firstName} ${lastName}`;**

Шаблонные строки — это способ создания строк с помощью обратных кавычек (`). Они появились в языке существенно позже, чем обычные строки и имеют в отличие от них ряд дополнительных возможностей.

1. Шаблонную строку, в отличие от обычной строки, можно располагать на нескольких строчках:
2. Для создания многострочных строк с помощью одинарных и двойных кавычек, нужно использовать символ перевода строки (\n): const mySkill = '\* HTML\n\* CSS\n\* JavaScript';
3. Расположить такую строку на нескольких строках можно только с использованием оператора +: const mySkill = '\* HTML\n' +

'\* CSS\n' +

'\* JavaScript';

1. Ещё одно отличие шаблонной строки заключается в том, что она можно включать выражения ${...}: const name = 'Alexander';

const greeting = `Hi, ${name.toUpperCase()}.`; // "Hi, ALEXANDER."

**2.Длина строки**

Получение длины строки, т.е. количества символов из которых она состоит, выполняется в JavaScript с помощью свойства length:

const mystr = 'Моя строка текста';

// получаем количество символов

**const len = mystr.length;**

**console.log(len); // 17**

**с**:

const name = 'Zoe';

// получим первый символ

**console.log(str[-1]); // undefined**

// получим второй символ

**console.log(str[3]); // undefined**

**Кроме этого получить** символ строки по его порядковому номеру можно с помощью специальных предназначенных для этого случая методов: **charAt и at.**

const str = 'Welcome';

// получим первый символ

**console.log(str.charAt[2]); // "l"**

// получим второй символ

**console.log(str.at[3]); // "c"**

Метод at появился в языке позже и в отличие от chartAt он более гибкий. Если ему передать на вход отрицательное число, то символ в этом случае будет отсчитываться с конца строки.

const fruit = 'Banana';

// получим последний символ

**console.log(fruit.at[-1]); // "a"**

**Если вам нужно посимвольно перебрать** **строку**, то можно воспользоваться циклами. Например, for или for..of:

**for (let char of "Hello") {**

**alert(char); // H,e,l,l,o (char — сначала "H", потом "e", потом "l" и т.д.)**

**3. Методы строк.**

**Методы для поиска подстроки в строке:**

**indexOf(substring [, start]) –** возвращает индекс, с которого начинается искомая подстрока в строке;

**lastIndexOf(substring, [, start]) –** то же самое, что indexOf, только поиск ведётся с конца строки.

**const vegetables = 'Tomato, Potato, Carrot, Tomato';**

**vegetables.indexOf('Tomato', 5); //** 24 (поиск (подстроки) Tomato начинается в 5 символа.

Параметр start является не обязательным. Он используется, когда нужно указать с какого символа строки необходимо начать поиск. Если его не передать методу, то поиск будет вестись с начала строки.

Если указанная подстрока не найдена, то эти методы в качестве результата возвращают значение -1.

В случае когда нужно проверить содержит ли указанная строка искомую, можно просто воспользоваться методом **includes:**

**const vegetables = 'Tomato, Potato, Carrot, Tomato';**

**vegetables.includes('Carrot'); // true -** Этот метод возвращает true, если искомая строка найдена. В противном случае, false.

Для поиска с определённой позиции, её необходимо просто передать во второй аргумент includes:

const file = 'index.html';

**file.includes**('.html', file.length - 5); // true

Проверить начало и конец строки на соответствие в JavaScript можно очень просто с помощью специальных методов **startsWith** и **endsWith:**

const href = 'https://itchief.ru/javascript/';

**href.startsWith**('https://itchief.ru'); // true

const file = 'index.html';

**file.endsWith**('.html'); // true

**Методы для получения подстроки:**

**substring(start [, end])** – часть строки между start и end (не включая);

const greeting = 'Hello, Bob! How are you?';

// если второй аргумент не указан, то до конца строки

greeting.substring(7); // "Bob! How are you?"

greeting.substring(7, 10); // "Bob"

**substr**(start [, length]) – часть строки, которая начинается с индекса start длиной length;

const greeting = 'Hello, Bob! How are you?';

// если второй аргумент не указан, то до конца строки

greeting.substr(7); // "Bob! How are you?"

// второй аргумент отвечает за количество возвращаемых символов

greeting.substr(7, 3); // "Bob"

// если первый аргумент отрицательный, то стартовый индекс отсчитывается с конца

greeting.substr(-17, 3); // "Bob"

**slice**(start [, end]) – часть строки между start и end (не включая);

// если второй аргумент не указан, то до конца строки

greeting.slice(7); // "Bob! How are you?"

// до end (не включая)

greeting.slice(7, 10); // "Bob"

// если start больше или равно end, то получим пустую строку

greeting.slice(10, 7); // ""

// если аргумент отрицательный, то индекс отсчитывается с конца

greeting.slice(-17, -14); // "Bob"

Разбиение строки на массив строк в JavaScript выполняется посредством метода **split:**

const nums = 'One,Two,Three';

// указатель для разделения строки передаётся в первый аргумент

nums.split(','); // ['One', 'Two', 'Three']

nums.split('One,'); // ['', 'Two,Three']

// ограничение на количество элементов, которые будут включены в массив, задаётся во втором аргументе

nums.split(',', 2); // ['One', 'Two']

// если указать в качестве разделителя пустую строку, то строка будет разделена посимвольно

nums.split('', 5); // ['O', 'n', 'e', ',', 'T']

Методы, с помощью которых можно в строке заменить **pattern на replacement**:

**replace**(pattern, replacement);

**replaceAll**(pattern, replacement).

В качестве pattern можно использовать строку или регулярное выражение, а для replacement – строку или функцию.

const nums = 'One, Two, Three, Two';

// заменим первую найденную строку "Two" на "Six"

nums.replace('Two', 'Six'); // One, Six, Three, Two

// заменим все найденные строки "Two" на "Six"

nums.replaceAll('Two', 'Six'); // One, Six, Three, Six

В этом примере с помощью метода replace заменили в nums подстроку 'Two' на 'Six'. Для этого мы в качестве первого аргумента передали в replace строку, которую нужно заменить. А во второй аргумент строку, на которую её нужно заменить. С помощью replaceAll мы заменили все вхождения 'Two' на'Six'.

Использование **replaceAll** для удаления пробелов из строки:

const str = 'A B C';

**const strWithoutSpaces = str.replaceAll(' ', '');**

console.log(strWithoutSpaces); // ABC

Если нужно убрать другой символ из строки:

**'Football'.replaceAll('o', ''); // "Ftball"**

Метод **repeat** позволяет создать строку, состояющую из указанного количества повторений другой:

'Bob! ' + 'Yes! '.repeat(3); // "Bob! Yes! Yes! Yes!"

Методы **toLowerCase и toUpperCase** предназначены для преобразования соответственно символов строки в строчные и прописные.

let myString = "Welcome to itchief!";

let upperCaseString = myString.toUpperCase();

let lowerCaseString = myString.toLowerCase();

console.log(upperCaseString); // "WELCOME TO ITCHIEF!"

console.log(lowerCaseString); // "welcome to itchief!"