**Строки**

Внутренний формат для строк — всегда [UTF-16](https://ru.wikipedia.org/wiki/UTF-16), вне зависимости от кодировки страницы.

**Кавычки**

3 типа - одинарные, двойные, обратные.

Одинарные и двойные кавычки работают, одинаково.

Обратные позволяют содавать Шаблонные строки

Возможности шаблонных строк:

* можно вставлять произвольные выражения, обернув их в ${…}
* в обратных кавычках содержимое может занимать более одной строки:

let guestList = `Guests:

\* John

\* Pete

\* Mary

`;

alert(guestList); // список гостей, состоящий из нескольких строк

если использовать в одинарных или двойных – будет ошибка.

let guestList = "Guests: // Error: Unexpected token ILLEGAL

\* John";

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

Обратные кавычки также позволяют задавать «шаблонную функцию» перед первой обратной кавычкой. Используемый синтаксис: func`string`. Автоматически вызываемая функция func получает строку и встроенные в неё выражения и может их обработать. Подробнее об этом можно прочитать в [документации](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/template_strings#%D0%A2%D0%B5%D0%B3%D0%BE%D0%B2%D1%8B%D0%B5_%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD%D1%8B). Если перед строкой есть выражение, то шаблонная строка называется «теговым шаблоном». Это позволяет использовать свою шаблонизацию для строк, но на практике теговые шаблоны применяются редко.

**Спецсимволы**

Многострочные строки также можно создавать с помощью одинарных и двойных кавычек, используя «символ перевода строки», который записывается как \n:

В частности, эти две строки эквивалентны, просто записаны по-разному:

// перевод строки добавлен с помощью символа перевода строки

let str1 = "Hello\nWorld";

// многострочная строка, созданная с использованием обратных кавычек

let str2 = `Hello

World`;

alert(str1 == str2); // true

Также если в одной строке добавить еще один пробел, а во второй строке оставить один пробел, результат сравнения будет false.

Некоторые спецсимволы

|  |  |
| --- | --- |
| \\ | Обратный слеш |
| \t | Знак табуляции |
| \', \", \` | Кавычки |

 все спецсимволы начинаются с обратного слеша, \ — «символа экранирования».

\n — считается как 1 символ! Важно для опдсчета длины строки.

**Доступ к символам**

1. через квдратные скобки, указав позицию символа

let str = `Hello`;

alert( str[0] ); // H

alert( str[str.length - 1] ); // o

Квадратные скобки всегда возвращают undefined для отрицательных индексов.

2. метод [str.at(pos)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/at). преимущество метода .at(pos) в том, что он допускает отрицательную позицию. Если pos – отрицательное число, то отсчет ведется от конца строки. Таким образом, .at(-1) означает последний символ, а .at(-2) – тот, что перед ним, и т.д.

let str = `Hello`;

alert( str.at(0) ); // H

alert( str.at(-1) ); // o

3. перебрать строку посимвольно, используя for..of:

for (let char of "Hello") {

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

}

**Строки неизменяемы**

Содержимое строки в JavaScript нельзя изменить. Нельзя взять символ посередине и заменить его. Как только строка создана — она такая навсегда.

let str = 'Hi';

str[0] = 'h'; //в «use strict» выпадет с ошибкой, в нестрогом, не приведет к ошибке, просто проигнорируется

alert( str ); // выведет первоначальное 'Hi'

**Изменение регистра**

Методы [toLowerCase()](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/String/toLowerCase) и [toUpperCase()](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/String/toUpperCase) меняют регистр символов. Могут применяться как к переменной, так и напрямую к строке.

Например перевести в нижний регистр какой-то конкретный символ:

alert( 'Interface'[0].toLowerCase() ); // 'i'

**Поиск подстроки**

**str.indexOf(substr, pos)**

Ищет подстроку **substr** в строке str, начиная с позиции pos (включая эту позицию) – необязательный аргумент.

Возвращает позицию, на которой располагается совпадение, либо

-1 при отсутствии совпадений. Поиск чувствителен к регистру.

let str = 'Widget with id';

alert( str.indexOf('Widget') ); // 0, потому что подстрока 'Widget' найдена в начале

alert( str.indexOf('ooooo') ); // -1, совпадений нет

alert( str.indexOf("id") ); // 1, подстрока "id" найдена на позиции 1

Чтобы найти все вхождения подстроки, нужно запустить indexOf в цикле. Каждый раз, получив очередную позицию, начинаем новый поиск со следующей:

let str = 'Ослик Иа-Иа посмотрел на виадук';

let target = 'Иа'; // цель поиска

let pos = 0;

while (true) {

let foundPos = str.indexOf(target, pos);

if (foundPos == -1) break;

alert( `Найдено тут: ${foundPos}` );

pos = foundPos + 1; // продолжаем со следующей позиции

}

Тот же алгоритм можно записать и короче:

let str = "Ослик Иа-Иа посмотрел на виадук";

let target = "Иа";

let pos = -1;

while ((pos = str.indexOf(target, pos + 1)) != -1) {

alert( pos );

}

При проверке indexOf в условии if надо делать всегда проверку на -1:

let str = "Widget with id";

if (str.indexOf("Widget") != -1) {

alert("Совпадение есть"); // теперь работает

}

Иначе, если подстрока будет найдена на индексе 0 , if не сработает.

let str = "Widget with id";

if (str.indexOf("Widget")) {

alert("Совпадение есть"); // не работает

}

**str.lastIndexOf(substr, position)**

Ищет с конца строки к её началу. Индексы возвращает только положительные. Строку ищет так как написано слева направо, не в зеркальном отражении))

Он используется тогда, когда нужно получить самое последнее вхождение: перед концом строки или начинающееся до (включительно) определённой позиции указанной во втором аргументе.

**Трюк с побитовым НЕ**

~n равно -(n+1)

В частности:

alert( ~0 ); // -1, то же, что -(0+1)

alert( ~-1 ); // 0, то же, что -(-1+1)

Таким образом, ~n равняется 0 только при n == -1, а также 0 преобразуется в -1 что в проверке if тоже даст true, в случае нахождения совпадения на индексе 0.

Соответственно, прохождение проверки if ( ~str.indexOf("…") ) означает, что результат indexOf отличен от -1, совпадение есть.

let str = "Widget";

if (~str.indexOf("Widget")) {

alert( 'Совпадение есть' ); // работает

}

На данный момент такой трюк можно встретить только в старом коде

это иногда применяют, чтобы сделать проверку indexOf компактнее.

Обычно использовать возможности языка каким-либо неочевидным образом не рекомендуется, но этот трюк широко используется в старом коде, поэтому его важно понимать. Сейчас для этого использут .includes

**str.includes(substr, pos)**

Это — правильный выбор, если необходимо проверить, есть ли совпадение, но позиция не нужна.

Возвращает true, если в строке str есть подстрока substr, либо false, если нет. Необязательный второй аргумент str.includes позволяет начать поиск с определённой позиции(включая указанную позицию).

alert( "Midget".includes("id") ); // true

alert( "Midget".includes("id", 1) ); // true, поиск начат с позиции 1 включительно

**str.startsWith и str.endsWith**

Проверяют, начинается ли и заканчивается ли строка определённой строкой:

alert( "Widget".startsWith("Wid") ); // true, "Wid" — начало "Widget"

alert( "Widget".endsWith("get") ); // true, "get" — окончание "Widget"

**Получение подстроки**

В JavaScript есть 3 метода для получения подстроки: substring, substr и slice.

**str.slice(start [, end])**

Возвращает часть строки от start(включая) до (не включая) end.

let str = "stringify";

// 's', от 0 до 1, не включая 1, т. е. только один символ на позиции 0

alert( str.slice(0, 1) );

Если аргумент end отсутствует, slice возвращает символы до конца строки:

let str = "stringify";

alert( str.slice(2) ); // ringify, с позиции 2 и до конца

Для start/end можно задавать отрицательные значения. Это означает, что позиция определена как заданное количество символов с конца строки, причем -1 это последний символ:

let str = "stringify";

// начинаем с позиции 4 справа, а заканчиваем на позиции 1 справа

alert( str.slice(-4, -1) ); // gif

**str.substring(start [, end])**

Возвращает часть строки между start и end (не включая) end.

Отличие от slice:

1.  можно задавать start больше end.Если start больше end, то метод substring сработает так, как если бы аргументы были поменяны местами.

let str = "stringify";

// для substring эти два примера — одинаковы

alert( str.substring(2, 6) ); // "ring"

alert( str.substring(6, 2) ); // "ring"

// …но не для slice:

alert( str.slice(2, 6) ); // "ring" (то же самое)

alert( str.slice(6, 2) ); // "" (пустая строка)

2. Отрицательные значения substring, в отличие от slice, не поддерживает, они интерпретируются как 0.

**str.substr(start [, length])**

Возвращает часть строки от start длины length. Второй арнумент необязательный, возвращает часть до конца строки.

В противоположность предыдущим методам, этот позволяет указать длину вместо конечной позиции:

let str = "stringify";

// ring, получаем 4 символа, начиная с позиции 2

alert( str.substr(2, 4) );

Значение первого аргумента может быть отрицательным, тогда позиция определяется с конца:

let str = "stringify";

// gi, получаем 2 символа, начиная с позиции 4 с конца строки

alert( str.substr(-4, 2) );

**Какой метод выбрать?**

Все эти методы эффективно выполняют задачу. Формально у метода substr есть небольшой недостаток: он описан не в собственно спецификации JavaScript, а в приложении к ней — Annex B. Это приложение описывает возможности языка для использования в браузерах, существующие в основном по историческим причинам. Таким образом, в другом окружении, отличном от браузера, он может не поддерживаться. Однако на практике он работает везде.

Из двух других вариантов, slice более гибок, он поддерживает отрицательные аргументы, и его короче писать. Так что, в принципе, можно запомнить только его.