

Цели занятия

- научиться создавать строки разными способами;
- изучить основные методы для работы со строками;
- разобраться с тем, как работает ветвление;

Полезные материалы

- [Подробная статья про все нюансы строк](#)
- [Полный справочник методов строк](#)
- [Подробно про конструкцию switch](#)

Строки

В JavaScript любые текстовые данные являются строками. Не существует отдельного типа «символ», который есть в ряде других языков. Для того чтобы создать строку в JavaScript можно использовать три типа кавычек: одинарные, двойные и обратные

```
let single = 'Какая-то строка';  
let double = "Вторая строка";  
let backticks = `И ещё одна строка`;
```

Одинарные и двойные кавычки работают, по сути, одинаково, а если использовать обратные кавычки, то в такую строку мы сможем вставлять произвольные выражения, обернув их в `${...}`:

```
let name = 'Юрий';  
let age = 26;  
  
let message = `Меня зовут ${name}, мне ${age} лет!`;  
// Меня зовут Юрий, мне 26 лет!
```

Ещё одно отличие обратных кавычек — они могут занимать более одной строки. А если попытаться сделать подобное с другими кавычками, то мы получим ошибку:

```
let list = `Список покупок:
* Картофель, 1 кг
* Морковь, 0.5кг
* Курица, 1шт
`;

console.log(list);
```

Справедливости ради стоит сказать, что с помощью обычных кавычек это тоже можно реализовать, но придётся вставлять специальный “символ перевода строки”, который записывается вот так `\n`

```
let list = "Список покупок: \n * Картофель, 1 кг \n * Морковь,
0.5кг \n * Курица, 1шт";

console.log(list);

// То же самое что и в примере выше,
// но гораздо хуже для понимания
```

Разница между методами и свойствами

В JavaScript есть такие понятие, как методы и свойства объекта. И для дальнейшего понимания материала необходимо понять чем они отличаются.

Методы нужно вызывать с помощью круглых скобок и они могут иметь дополнительные параметры, а свойства сразу возвращают значение (причём это относится не только к строкам, а вообще ко всему языку).

Например:

```
let num = 1.245;
let roundedNum = num.toFixed(1); // Это метод, он принимает
параметр

let message = 'Hello, Frontenders!';
let l = message.length; // А это свойство (длина строки)

console.log(l); // 19
```

Доступ к отдельным символам строки

Получить символ на определенной позиции можно с помощью квадратных скобок [], либо с помощью метода charAt(). Первый способ более современный, а второй существует по историческим причинам. Отдельно стоит обратить внимание, что позиция начинается с нуля.

```
let str = `Hello`;

// получаем первый символ
alert( str[0] ); // H
alert( str.charAt(0) ); // H

// получаем последний символ
alert( str[str.length - 1] ); // o
```

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'; // Вот здесь будет ошибка
alert( str ); // Не работает
```

Можно создать новую строку и записать её в ту же самую переменную вместо старой:

```
let str = 'Hi';

str = 'h' + str[1]; // Заменяем строку
alert( str ); // hi
```

Изменение регистра строки

Методы `toLowerCase()` и `toUpperCase()` меняют регистр символов во всей строке. Проще всего понять принцип их работы на конкретных примерах:

```
let message = 'Hello, FroNtenDers!';

alert( message.toUpperCase() ); // HELLO, FRONTENDERS!
alert( message.toLowerCase() ); // hello, frontenders!
```

Поиск внутри строки

Одна из самых частных потребностей при работе со строками это проверить, что в строке есть необходимые нам части или строка начинается/заканчивается определенными символами. Для этого у строк есть методы `includes()`, `startsWith()`, `endsWith()`

```
let message = 'Hello, frontenders!';

// В строке есть указанная подстрока?
alert( message.includes("frontenders") ); // true
alert( message.includes("backenders") ); // false

// Строка начинается на "Hello" ?
alert( message.startsWith("Hello") ); // true

// Строка заканчивается на "!" ?
alert( message.endsWith("!") ); // true

// Строка заканчивается на "test" ?
alert( message.endsWith("test") ); // false
```

Кроме этого существует ещё и метод [indexOf\(\)](#), но используется он реже, поэтому его изучение выходит за рамки занятия.

Другие полезные методы строк

- **slice(start, end)** — Возвращает часть строки от `start` до (не включая) `end`. Если аргумент `end` отсутствует, `slice` возвращает символы до конца строки

```
let str = "JavaScript";

// от 0 до 4, то есть "Java"
alert( str.slice(0, 4) );
```

```
// от 0 до 1, не включая 1, то есть "J"
alert( str.slice(0, 1) );

// от 3 до конца строки, то есть "Script"
alert( str.slice(4) );
```

- **trim()** — удаляет пробельные символы с начала и конца строки. Пробельными символами в этом контексте считаются все собственно пробельные символы (пробел, табуляция, неразрывный пробел и прочие) и все символы конца строки

```
let message = ' Hello, Frontenders! ';

// Удалит все пробелы в начале и конце
alert( message.trim() ); // "Hello, Frontenders!"

// Удалит пробелы только в начале
alert( message.trimStart() ); // "Hello, Frontenders! "

// Удалит пробелы только в конце
alert( message.trimEnd() ); // " Hello, Frontenders!"
```

- **replace(str, newStr)** — возвращает новую строку в которой указанная часть была заменена на заменитель

```
let message = 'Я знаю HTML, CSS и PHP!';

alert( message.replace('PHP', 'JavaScript') );
// Я знаю HTML, CSS и JavaScript!
```

Взаимодействие с пользователем

В следующей теме нам нужно будет получать информацию от пользователя, поэтому давайте познакомимся с некоторыми простыми функциями для этого:

- **alert(message)** — с этой функцией мы уже знакомы. Она показывает сообщение и ждёт, пока пользователь нажмёт кнопку «ОК».

```
alert("Hello");
```

Это небольшое окно с сообщением называется *модальным окном*. Понятие *модальное* означает, что пользователь не может взаимодействовать с интерфейсом остальной части страницы, нажимать на другие кнопки и т.д. до тех пор, пока взаимодействует с окном. В данном случае – пока не будет нажата кнопка «ОК».

- **prompt(message)** — позволяет узнать у пользователя какую-то информацию, например, попросить ввести своё имя. Значение, которое ввёл пользователь можно сохранить в переменную и использовать в своём коде, например:

```
let name = prompt('Как вас зовут?');  
  
alert(`О! Привет, ${name}!`); // О! Привет, Юрий!
```

- **confirm(question)** — отображает модальное окно с текстом вопроса question и двумя кнопками: ОК и Отмена. Результат – true, если нажата кнопка ОК. В других случаях – false.

```
let isBoss = confirm("Ты здесь главный?");  
  
alert( isBoss ); // true, если нажата ОК
```

Условное ветвление

Иногда нам нужно выполнить различные действия в зависимости от условий. Для этого мы можем использовать инструкцию `if ... else`

С помощью `if`

Инструкция `if(...)` вычисляет условие в скобках и, если результат `true`, то выполняет блок кода.

```
let year = prompt('В каком году была опубликована спецификация ECMAScript-2015?');

// Всё что внутри фигурных скобок выполнится
// только при правильном ответе
if (year === 2015) {
    alert( "Правильно!" );
    alert( "Вы такой умный!" );
}
```

Инструкция `if` может содержать необязательный блок «else» («иначе»). Он выполняется, когда условие ложно.

```
let age = prompt('Сколько вам лет?');

if (age < 18) {
    alert( 'Вы ещё так юны, немедленно покиньте страницу!' );
} else {
    // любое значение, которое больше или равно 18
    alert( 'Всё в порядке, можно смотреть :)' );
}
```

С помощью оператора ?

Иногда нам нужно определить переменную в зависимости от условия.

```
let message;
let age = prompt('Сколько вам лет?');

if (age < 18) {
    message = 'Вы ещё так юны, немедленно покиньте страницу!';
} else {
    message = 'Всё в порядке, можно смотреть :)';
}

alert(message);
```

Так называемый «условный» оператор «вопросительный знак» позволяет нам сделать это более коротким и простым способом.

Оператор представлен знаком вопроса ?. Его также называют «тернарный», так как этот оператор, единственный в своём роде, имеет три аргумента.

```
let result = условие ? значение1 : значение2;
```

Сначала вычисляется условие: если оно истинно, тогда возвращается значение1, в противном случае – значение2.

Теперь мы можем переписать пример выше, но сократить количество кода с помощью тернарного оператора:

```
let age = prompt('Сколько вам лет?');
let msg = age < 18 ? 'Покиньте страницу!' : 'Всё в порядке!';

alert(msg);
```

Конструкция switch

Конструкция switch заменяет собой сразу несколько if. Она представляет собой более наглядный способ сравнить выражение сразу с несколькими вариантами.

```
let answer = prompt('Сколько будет 2+2?');

switch (answer) {
    // Выполнится только если answer равен 3
    case 3:
        alert( 'Маловато' );
        break;

    // Выполнится только если answer равен 4
    case 4:
        alert( 'В точку!' );
        break;

    // Выполнится только если answer равен 5
    case 5:
        alert( 'Перебор' );
        break;

    // Выполнится когда answer не попал ни в одно условие
    default:
        alert( "Нет таких значений" );
}
```

Здесь оператор switch последовательно сравнит `a` со всеми вариантами из `case`.

Сначала 3, затем – так как нет совпадения – 4. Совпадение найдено, будет выполнен этот вариант, со строки `alert( 'В точку!' )` и далее, до ближайшего `break`, который прервёт выполнение.

Если `break` нет, то выполнение пойдёт ниже по следующим `case`, при этом остальные проверки игнорируются.