Основы JavaScript

- JavaScript
- Цель
  - Основы JavaScript
    - Типы данных
      - number
      - string
      - boolean
      - null
      - undefined
      - Object
      - Symbol
    - Операторы
    - Условия
    - Циклы
    - Функции
      - Классификация
      - Замыкание функций
      - callback-функции
    - ООП
    - Динамическая типизация
  - Работа со страницей
    - Получение элементов
    - События
      - Всплытие событий
      - Делегирование событий
      - Объект события
      - Цикл события
    - Обработчики событий
    - Отмена стандартных событий браузера Ajax
    - Параметры глобальных объектов
      - window (видимая часть браузера)
      - screen (весь ваш экран)
      - document
    - js параметры элементов (геометрия)
    - Работа скриптов с течением времени
  - Регулярные выражения
  - JS в работе
    - this Контекст вызова
    - Конструкторы
    - Формат передачи данных JSON
    - Общение с сервером, AJAX
    - Инкапсуляция

Сборка проекта
webpack
browserify
...
Перехват ошибок try/catch
работа с LocalStorage

Типы данных

- number
- string
- boolean
- null
- undefined
- Object
- Symbol

Их характеристики:

1. Number (Число)

- Представляет как целые числа, так и числа с плавающей точкой
- Диапазон от -(2^53-1) до (2^53-1)
- Специальные значения: Infinity, -Infinity и NaN
- **Пример:**

```
let age = 25;
let price = 99.99;
```

2. String (Строка)

- Представляет текстовые данные
- Может быть заключена в одинарные ('), двойные ("") или обратные (`) кавычки
- Поддерживает экранирование специальных символов
- **Пример:**

```
let name = "Анна";
let greeting = Привет, ${name}!;
```

3. Boolean (Логический тип)

- Имеет только два значения: true и false
- Используется для логических операций
- **Пример:**

```
let isOnline = true;
let isAdmin = false;
```

4. Null

- Специальное значение, представляющее "ничего" или "пусто"
- Является отдельным типом данных
- **Пример:**

```
let user = null;
```

5. Undefined

- Означает, что значение не было присвоено
- Автоматически присваивается переменным при объявлении без инициализации
- **Пример:**

```
let x; // x имеет значение undefined
```

6. Object (Объект)

- Коллекция пар ключ-значение
- Может содержать свойства и методы
- Включает массивы, функции и другие сложные структуры данных
- **Пример:**

```
let person = {
    name: "Иван",
    age: 30,
    isStudent: false
};
```

7. Symbol (Символ)

- Уникальный и неизменяемый тип данных
- Часто используется как ключ для свойств объекта
- Создается с помощью функции Symbol()
- Каждый символ уникален, даже если создан с одинаковым описанием
- **Пример:**

```
let id = Symbol("id");
```

Важные характеристики всех типов данных в JavaScript:

- Все типы кроме Object являются примитивными
- Примитивные типы иммутабельны (неизменяемы)
- JavaScript является динамически типизированным языком
- Все примитивы передаются по значению, объекты - по ссылке
- Для проверки типа можно использовать оператор typeof

```
let person = {
        name: "Иван",
        age: 30,
        isStudent: false
        };
```

Как работать с объектом person различными способами:

```
// 1. Доступ к свойствам через точку
let person = {
    name: "Иван",
    age: 30,
    isStudent: false
```

```
};

// Чтение свойств
console.log(person.name);       // "Иван"
console.log(person.age);        // 30
console.log(person.isStudent);  // false

// 2. Доступ к свойствам через квадратные скобки
console.log(person["name"]);   // "Иван"

// 3. Изменение значений свойств
person.age = 31;
person["name"] = "Петр";

// 4. Добавление новых свойств
person.city = "Москва";
person["email"] = "ivan@example.com";

// 5. Удаление свойств
delete person.isStudent;

// 6. Проверка существования свойства
console.log("name" in person);       // true
console.log(person.hasOwnProperty("age")); // true

// 7. Перебор всех свойств объекта
for (let key in person) {
    console.log(`${key}: ${person[key]}`);
}

// 8. Получение массива ключей
let keys = Object.keys(person);   // ["name", "age", "city", "email"]

// 9. Получение массива значений
let values = Object.values(person);

// 10. Создание копии объекта
let personCopy = Object.assign({}, person);
// или через spread-оператор
let anotherCopy = { ...person };

// 11. Вложенные объекты
person.address = {
    city: "Москва",
    street: "Ленина"
```

```
};
console.log(person.address.city); // "Москва"

// 12. Деструктуризация
let { name, age } = person;
console.log(name); // "Петр"
console.log(age);  // 31
```

Основные моменты при работе с объектами:

1. Свойства объекта можно получать двумя способами:
   - Через точку (person.name)
   - Через квадратные скобки (person["name"])
2. Квадратные скобки полезны когда:
   - Имя свойства хранится в переменной
   - Имя свойства содержит пробелы или специальные символы
3. Объекты можно:
   - Изменять существующие свойства
   - Добавлять новые свойства
   - Удалять свойства через delete
   - Проверять наличие свойств через in или hasOwnProperty()
   - Перебирать через цикл for...in
4. При работе с вложенными объектами нужно учитывать:
   - Доступ к свойствам через цепочку точек
   - При копировании нужно использовать глубокое копирование для вложенных объектов
5. Методы объекта:
   - Object.keys() - получить массив ключей
   - Object.values() - получить массив значений
   - Object.entries() - получить массив пар [ключ, значение]
6. Деструктуризация позволяет удобно извлекать свойства в отдельные переменные

1. Получение свойств двумя способами:

```
let user = { name: "Анна", age: 25 };
// Через точку
console.log(user.name);       // "Анна"
console.log(user.age);        // 25
// Через скобки
console.log(user["name"]);    // "Анна"
console.log(user["age"]);     // 25
```

2. Использование квадратных скобок:

```
// С переменной
let propName = "name";
console.log(user[propName]);  // "Анна"

// Со специальными символами
```

```
let person = {
    "first name": "Иван",
    "birth date": "01.01.1990"
};
console.log(person["first name"]);    // "Иван"
console.log(person["birth date"]);    // "01.01.1990"
```

3. Операции с объектами:

```
// Изменение и добавление свойств
let car = { brand: "Toyota" };
car.brand = "Honda";                  // изменение
car.model = "Civic";                  // добавление

// Удаление и проверка
let computer = { cpu: "Intel", ram: "16GB" };
delete computer.ram;
console.log("cpu" in computer);       // true
console.log(computer.hasOwnProperty("ram")); // false
```

4. Вложенные объекты:

```
// Доступ через цепочку
let company = {
    department: {
        name: "IT",
        manager: "Петр"
    }
};
console.log(company.department.name);  // "IT"

// Глубокое копирование
let original = { a: { b: 2 } };
let deepCopy = JSON.parse(JSON.stringify(original));
```

5. Методы объекта:

```
// Keys и Values
let phone = { brand: "Apple", model: "iPhone", year: 2023 };
console.log(Object.keys(phone));      // ["brand", "model", "year"]
console.log(Object.values(phone));    // ["Apple", "iPhone", 2023]

// Entries
let laptop = { brand: "Dell", price: 1000 };
console.log(Object.entries(laptop));   // [["brand", "Dell"], ["price", 1000]]
```

6. Деструктуризация:

```
// Базовая деструктуризация
let student = { name: "Мария", grade: "A", age: 20 };
let { name, grade } = student;
console.log(name, grade);             // "Мария" "A"
```

```
// С переименованием и значениями по умолчанию
let book = { title: "JavaScript" };
let { title: bookName, author = "Unknown" } = book;
console.log(bookName, author);       // "JavaScript" "Unknown"
```

Язык JavaScript предназначен для выполнения в браузере наряду с HTML и CSS. Но, если эти языки предназначены для верстки структуры сайта, то JavaScript позволяет "оживлять" web-страницы - делать их реагирующими на действия пользователя или демонстрировать некоторую динамичность (к примеру, смена картинок в блоке или красивые плавно выпадающие менюшки).

Запуск JavaScript

JavaScript код пишется прямо на HTML странице внутри тега script. Этот тег можно размещать в любом месте страницы. Смотрите пример:

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset="utf-8">
		<title>Это заголовок тайтл</title>
		<script>
			здесь пишем JavaScript код
		</script>
	</head>
	<body>
		Это основное содержимое страницы.
	</body>
</html>
```

Первая программа на JavaScript

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset="utf-8">
		<title>Моя первая программа</title>
		<script>
			alert('Привет, мир!');
		</script>
	</head>
	<body>
		моя первая программа
	</body>
</html>
```

Скопируйте этот код и разместите в HTML файле. Затем откройте этот файл в браузере - и вы увидите диалоговое окошко с текстом.

Первым понятием, которое вам нужно узнать, являются *функции*. Функции позволяют выполнять некоторые действия. В нашем примере есть функция alert(), которая выводит текст на экран в виде диалогового окошка.

Функция состоит из имени (в нашем случае это alert) и круглых скобок, написанных после этого имени. В этих круглых скобках следует писать *параметры функции*. В нашем случае параметром является текст, который выводится на экран.

В нашем случае функция имеет один параметр, однако бывают функции, в которые нужно передавать несколько параметров. В этом случае эти параметры пишутся через запятую.

Далее не будет расписываться то, как подключается JavaScript, а будет просто написано JavaScript код, подразумевая, что вы его будете записывать в тегах script. С учетом этого замечания наша программа станет выглядеть вот так:

```
alert('Привет, мир!');
```

Подключение файла со скриптами JavaScript

JavaScript код можно писать в отдельном файле, который затем будет подключен к HTML файлу. Давайте посмотрим, как это делается.

Для начала создадим файл с нашим скриптом. У этого файла должно быть расширение .js. Для примера давайте назовем его script.js. Разместим в нем какой-нибудь код:

```
alert('text');
```

Давайте теперь подключим наш файл со скриптом к HTML файлу. Для этого в теге script в атрибуте src нужно указать путь к файлу со скриптом:

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset="utf-8">
		<title></title>
		<script src="js/script.js"></script>
	</head>
	<body>

	</body>
</html>
```

Несколько файлов со скриптами JavaScript

Можно подключать не один, а несколько файлов с помощью нескольких тегов script:

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset="utf-8">
		<title></title>
		<script src="script1.js"></script>
		<script src="script2.js"></script>
	</head>
	<body>

	</body>
</html>
```

Особенности тега script в JavaScript

Учтите, что в теге script можно либо писать код, либо подключить файл. Попытка сделать это одновременно не будет работать. Поэтому следующий код не рабочий:

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset="utf-8">
		<title></title>
		<script src="script.js">
			alert('text');
		</script>
	</head>
	<body>

	</body>
</html>
```

Кеширование JavaScript файлов браузером

Браузер может кешировать подключенные JavaScript файлы. Это значит, что он сохраняет их у себя для повышения скорости загрузки сайта.

На практике это ведет к тому, что в какой-то момент при редактировании кода браузер будет применять предыдущую версию кода, а не вашу текущую.

Для борьбы с таким поведением можно чистить кеш браузера, либо использовать хитрый прием. Его суть заключается в том, что мы при подключении скрипта после имени файла ставим вопрос, знак равно и номер

версии вашего скрипта. Когда браузер кеширует файл, вам просто нужно будет увеличить номер на единицу.

Пример:

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset="utf-8">
		<title></title>
		<script src="script.js?v=1"></script>
	</head>
	<body>

	</body>
</html>
```

Строгий режим в JavaScript

В современном JavaScript при написании кода первой строчкой нужно включать так называемый *строгий режим*. Этот режим заставляет браузер использовать все современные возможности языка.

Для включения строго режима первой строчкой скрипта необходимо поставить команду "use strict". Давайте перепишем наш код в строгом режиме:

```
"use strict";
alert('text!');
```

В дальнейшем для простоты указывается то, что код работает в строгом режиме (но подразумевается, что да, и вы всегда его включайте).

Включите в вашем скрипте строгий режим.

Комментарии в JavaScript

Подобно языкам HTML и CSS в языке JavaScript можно оставлять комментарии. Они бывают однострочными и многострочными.

Вот пример однострочного комментария:

```
alert('Привет, мир!'); // комментарий
```

Вот пример многострочного комментария:

```
/*
	комментарий
*/
alert('Привет, мир!');
```

Попробуйте оба типа комментариев.

Переменные в JavaScript

Основным понятием любого языка программирования является переменная. Переменная представляет собой контейнер, в котором мы можем хранить какие-либо данные, например, строки или числа.

Каждая переменная должна иметь имя, которое может состоять из латинских букв, чисел, символов $ и знаков подчеркивания. При этом первый символ имени переменной не должен быть цифрой. Примеры

названий переменных: str, my_str, myStr, a1, $, $a, $$a.

Использование переменных

Для того, чтобы использовать переменную, ее сначала необходимо *объявить*: написать перед ее именем ключевое слово let. Давайте объявим, например, переменную с именем a:

```
let a;
```

После объявления переменной в нее можно записать (говорят *присвоить* ей) какое-либо значение, например, какое-либо число или строку.

Запись данных в переменную осуществляется с помощью операции присваивания =. Давайте, например, запишем в переменную a число 3:

```
let a = 3;
```

А теперь давайте выведем содержимое этой переменной на экран с помощью функции alert:

```
let a = 3; // объявляем переменную и задаем ей значение
alert(a);  // выведет 3
```

Не обязательно записывать значение в переменную сразу после объявления. Можно сначала объявить переменную, а потом присвоить ей значение:

```
let a;    // объявим переменную
a = 3;    // присвоим ей значение
alert(a); // выведем значение переменной на экран
```

Как вы видите, let перед именем переменной пишется только один раз - при объявлении этой переменной. Затем, чтобы использовать переменную, нужно просто писать имя этой переменной.

Объявление нескольких переменных в JavaScript

Давайте объявим несколько переменных:

```
let a = 1;
let b = 2;
let c = 3;
```

Приведенный выше код можно упростить, написав let один раз и после него перечислив нужные переменные с их значениями, вот так:

```
let a = 1, b = 2, c = 3;
```

Можно вначале объявить все переменные, а потом присваивать им значения:

```
let a, b, c; // объявляем все 3 переменные

// Присваиваем переменным значения:
a = 1;
b = 2;
c = 3;
```

Самостоятельно опробуйте все описанные способы объявления переменных.

Изменения значений переменных

Пусть у нас есть переменная, которой мы присвоили какое-то значение. Мы затем можем выполнить операцию присваивания еще раз и присвоить переменной другое значение:

```
let a;    // объявляем переменную

a = 1;    // записываем в нее значение 1
alert(a); // выведет 1

a = 2;    // записываем теперь значение 2, затирая значение 1
alert(a); // выведет 2
```

То есть значение переменной не является чем-то жестко привязанным к ней. Мы можем свободно записывать какие-то данные в переменную, прочитывать их, потом еще что-то записывать - и так далее.

Несколько let для одной переменной

Одну и ту же переменную нельзя объявить несколько раз через let. К примеру, следующий код приведет к ошибке:

```
let a = 1;
alert(a);

let a = 2;
alert(a);
```

Здесь есть два варианта решения проблемы. Можно просто ввести две разных переменных:

```
let a = 1;
alert(a);
```

```
let b = 2;
alert(b);
```

А можно вначале объявить переменную a, а затем делать операции с нею:

```
let a;
a = 1;
alert(a);
a = 2;
alert(a);
```

Математические операции с числами в JavaScript

Сложение:

```
let a = 1 + 2;
alert(a); // выведет 3
```

Вычитание:

```
let b = 3 - 2;
alert(b); // выведет 1
```

Умножение:

```
let c = 3 * 2;
alert(c); // выведет 6
```

Деление:

```
let d = 4 / 2;
alert(d); // выведет 2
```

Математические операции можно производить не только над числами, но и над переменными. Сложим, к примеру, значения двух переменных:

```
let a = 1;
let b = 2;
alert(a + b); // выведет 3
```

Не обязательно сразу выводить результат операции, можно вначале записать его в какую нибудь переменную, а уже затем вывести значение этой переменной:

```
let a = 1;
let b = 2;
let c = a + b; // запишем сумму в переменную c
alert(c);      // выведет 3
```

Приоритет математических операций в JavaScript

Математические операции JavaScript имеют такой же приоритет, как в обычной математике. То есть в начале выполняется умножение и деление, а потом уже сложение и вычитание.

В следующем примере вначале 2 умножится на 2 и затем к результату прибавится 3:

```
let a = 2 * 2 + 3;
alert(a); // выведет 7 (результат 4 + 3)
```

Не запуская код, определите, что выведется на экран:

```
let a = 5 + 5 * 3;
alert(a);
```

Не запуская код, определите, что выведется на экран:

```
let a = 5 + 5 * 3 + 3;
alert(a);
```

Не запуская код, определите, что выведется на экран:

```
let a = 8 / 2 + 2;
alert(a);
```

Не запуская код, определите, что выведется на экран:

```
let a = 8 + 2 / 2;
alert(a);
```

Равный приоритет математических операций в JavaScrip

Умножение и деление имеют равный приоритет и выполняются по очереди слева направо. Рассмотрим на примере, что имеется ввиду. В следующем коде вначале выполнится деление, а потом умножение:

```
let a = 8 / 2 * 4;
alert(a); // выведет 16 (результат 4 * 4)
```

Если же переставить знаки местами, то вначале выполнится умножение, а потом деление:

```
let a = 8 * 2 / 4;
alert(a); // выведет 4 (результат 16 / 4)
```

В следующем примере каждая новая операция деления будет применяться к предыдущей:

```
let a = 16 / 2 / 2 / 2;
alert(a); // выведет 2
```

Не запуская код, определите, что выведется на экран:

```
let a = 8 / 2 * 2;
```

```
alert(a);
```

Не запуская код, определите, что выведется на экран:

```
let a = 8 * 4 / 2 / 2;
alert(a);
```

Группирующие скобки в JavaScript

При желании вы можете указывать приоритет операций с помощью круглых скобок. Давайте, например, переделаем наш код так, чтобы вначале выполнилось сложение, а уже потом умножение:

```
let a = 2 * (2 + 3);
alert(a); // выведет 10 (результат 2 * 5)
```

Примечание: скобок может быть любое количество, в том числе и вложенных друг в друга:

```
let a = 2 * (2 + 4 * (3 + 1));
alert(a);
```

В скобки можно заключить операции, обладающие приоритетом - это не будет ошибкой. Например, заключим в скобки произведение чисел:

```
let a = (2 * 2) + 3;
alert(a); // выведет 7 (результат 4 + 3)
```

В данном случае скобки получаются лишними (у умножения ведь и так приоритет), но код является допустимым.

Иногда такую группировку используют в тех местах, где приоритет операций не очевиден. Для примера рассмотрим следующий код:

```
let a = 8 / 2 * 4;
alert(a);
```

Как вы уже знаете, в нем вначале выполнится деление, а потом умножение. Но с первого взгляда это может быть не слишком очевидно.

Можно использовать группирующие скобки, чтобы явно показать приоритет:

```
let a = (8 / 2) * 4;
alert(a);
```

Не запуская код, определите, что выведется на экран:

```
let a = (2 + 3) * (2 + 3);
alert(a);
```

Не запуская код, определите, что выведется на экран:

```
let a = (2 + 3) * 2 + 3;
alert(a);
```

Не запуская код, определите, что выведется на экран:

```
let a = 2 * (2 + 4 * (3 + 1));
alert(a);
```

Не запуская код, определите, что выведется на экран:

```
let a = 2 * 8 / 4;
alert(a);
```

Не запуская код, определите, что выведется на экран:

```
let a = (2 * 8) / 4;
alert(a);
```

Не запуская код, определите, что выведется на экран:

```
let a = 2 * (8 / 4);
alert(a);
```

Дроби в JavaScript

В JavaScript можно работать с десятичными дробями. При этом целая и дробная части отделяются друг от друга точкой. Смотрите пример:

```
let a = 0.5;
alert(a); // выведет 0.5
```

Смотрите еще пример:

```
let a = 0.5 + 0.5;
alert(a); // выведет 1
```

Запишите в переменную a число 1.5, а в переменную b - число 0.75. Найдите сумму значений этих переменных и выведите ее на экран.

Отрицательные числа в JavaScript

Числа могут быть отрицательными. Для этого перед числом необходимо поставить знак минус:

```
let a = -1;
alert(a); // выведет -1
```

Знак минус можно писать как к числам, так и к переменным:

```
let a = 1;
let b = -a; // записали в b содержимое a с обратным знаком
alert(b);   // выведет -1
```

Или вот так:

```
let a = 1;
alert(-a); // выведет -1
```

Плюс перед числами в JavaScript

Подобно тому, как перед отрицательными числами ставится знак "минус", перед положительными числами можно ставить знак "плюс".

Фактически этот плюс ничего не делает, но вполне является допустимым, смотрите пример:

```
let a = +1;
alert(a); // выведет 1
```

Остаток от деления в JavaScript

Существует специальный оператор %, с помощью которого можно найти остаток от деления одного числа на другое:

```
alert(10 % 3); // выведет 1
```

Если одно число делится нацело на второе - остаток будет равен нулю:

```
alert(10 % 2); // выведет 0
```

Оператор %, конечно же, можно применять не только к числам, но и к переменным:

```
let a = 10;
let b = 3;
alert(a % b); // выведет 1
```

Найдите остаток от деления следующих переменных:

```
let a = 13;
let b = 5;
```

Возведение в степень в JavaScript

Для возведения числа в степень также существует специальный оператор **. Давайте с его помощью возведем число 10 в третью степень:

```
alert(10 ** 3); // выведет 1000
```

Давайте возведем в степень значение переменной:

```
let a = 10;
alert(a ** 3); // выведет 1000
```

Может быть такое, что и число, и степень будут содержаться в переменных:

```
let a = 10;
let b = 3;
alert(a ** b); // выведет 1000
```

Приоритет возведения в степень

Операция возведения в степень имеет приоритет перед умножением и делением. В следующем примере вначале выполнится возведение в степень, а затем умножение:

```
alert(2 * 2 ** 3);
```

Не запуская код, определите, что выведется на экран:

```
let a = 3 * 2 ** 3;
alert(a);
```

Не запуская код, определите, что выведется на экран:

```
let a = (3 * 2) ** 3;
alert(a);
```

Не запуская код, определите, что выведется на экран:

```
let a = 3 * 2 ** (3 + 1);
alert(a);
```

Не запуская код, определите, что выведется на экран:

```
let a = 2 ** 3 * 3;
alert(a);
```

Не запуская код, определите, что выведется на экран:

```
let a = 3 * 2 ** 3 * 3;
alert(a);
```

Строки в JavaScript

Как уже упоминалось ранее, данные могут иметь различный тип. Один из типов - числа, мы уже немного изучили. Давайте теперь перейдем к строкам.

Строки создаются с помощью кавычек:

```
let str = 'abc';
alert(str); // выведет 'abc'
```

Кавычки могут быть не только одинарными, но и двойными:

```
let str = "abc";
alert(str); // выведет 'abc'
```

Между одинарными и двойными кавычками в JavaScript нет никакой разницы. Их использование зависит от ваших предпочтений. Я предпочитаю использовать одинарные кавычки, поэтому везде далее в учебнике будут стоять именно они.

Здесь и далее, если результат вывода - строка, то я беру его в кавычки, чтобы показать, что это именно строка, вот так: *выведет 'abc'*. При выводе

строки через alert никаких кавычек появляться не будет (то есть на экран выведется то, что у меня написано внутри кавычек).

Сложение строк в JavaScript

Для сложения строк, так же, как и для сложения чисел, используется оператор +:

```
let str = 'abc' + 'def'; // складываем две строки
alert(str); // выведет 'abcdef'
```

Строки также могут хранится в переменных:

```
let str1 = 'abc';
let str2 = 'def';
alert(str1 + str2); // выведет 'abcdef'
```

Можно также складывать переменные и строки:

```
let str1 = 'abc';
let str2 = 'def';
alert(str1 + '!!!' + str2); // выведет 'abc!!!def'
```

Пусть две строки хранятся в переменных, а при их сложении мы хотим вставить между ними пробел. Это делается так:

```
let str1 = 'abc';
let str2 = 'def';
alert(str1 + ' ' + str2); // выведет 'abc def'
```

Пусть переменная только одна:

```
let str = 'abc';
alert(str + ' ' + 'def'); // выведет 'abc def'
```

В этом случае нет смысла выделять пробел, как отдельную строку - вставим его как часть второго слагаемого:

```
let str = 'abc';
alert(str + ' def'); // выведет 'abc def'
```

Длина строки в JavaScript

Количество символов в строке содержится в свойстве length:

```
let str = 'abcde';
alert(str.length); // выведет 5
```

Свойство length можно применять непосредственно к строке:

```
alert('abcde'.length); // выведет 5
```

Пробел также является символом:

```
alert('ab de'.length); // выведет 5
```

Запишите в переменную какую-нибудь строку. Выведите на экран длину вашей строки.

Шаблонные строки в JavaScript

Существует специальный тип кавычек - косые:

```
let str = `abc`;
alert(str); // выведет 'abc'
```

В косых кавычках можно выполнять вставку переменных. Для этого имя переменной нужно написать в конструкции ${}.

Давайте посмотрим на примере. Пусть мы хотим выполнить сложение строк и переменной:

```
let str = 'xxx';
let txt = 'aaa ' + str + ' bbb';
```

Этот код можно переписать следующим образом:

```
let str = 'xxx';
let txt = `aaa ${str} bbb`;
```

Перепишите следующий код через вставку переменных:

```
let str1 = 'xxx';
let str2 = 'yyy';
let txt  = 'aaa ' + str1 + ' bbb ' + str2 + ' ccc';
```

Многострочность

В JavaScript строках, созданных через одинарные или двойные кавычки, не допустим перенос строки. То есть вот так не будет работать:

```
let str = 'abc
def'; // так не будет работать
```

И так не будет работать:

```
let str = "abc
def"; // так не будет работать
```

А вот косые кавычки специально предназначены для создания многострочных строк:

```
let str = `abc
def`; // так будет работать
```