

JS: Functions

НАШИ ПРАВИЛА



Включенная камера



Вопросы по поднятой руке



Не перебиваем друг друга



Все вопросы, не связанные с тематикой курса (орг-вопросы и т. д.), должны быть направлены куратору



Подготовьте свое рабочее окружение для возможной демонстрации экрана (закройте лишние соцсети и прочие приложения)

ПОИГРАЕМ ;)

Какие особенности массива в JS?

Что делают методы `push`, `shift`, `unshift`, `pop`?

Как вызвать определенный элемент массива

ЦЕЛЬ

Изучить циклы `while`, `do...while`. Изучить функции в JS

ПЛАН ЗАНЯТИЯ

- Циклы: while, do while
- Методы строк
- Functions: introduction
- Область видимости, hoisting (поднятие)

Циклы



Цикл while

Цикл while имеет следующий синтаксис:

```
1 while (condition) {  
2     // код  
3     // также называемый "телом цикла"  
4 }
```

Цикл while

Код из тела цикла выполняется, пока условие condition истинно.

Например, цикл ниже выводит *i*, пока $i < 3$:

```
1 let i = 0;
2 while (i < 3) { // выводит 0, затем 1, затем 2
3   alert( i );
4   i++;
5 }
```


Цикл while

Любое выражение или переменная может быть условием цикла, а не только сравнение: условие `while` вычисляется и преобразуется в логическое значение.

Например, `while (i)` – более краткий вариант `while (i != 0)`:

```
1  let i = 3;
2  while (i) { // когда i будет равно 0, условие станет ложным
3      alert( i );
4      i--;
5  }
```

Цикл “do...while”

Проверку условия можно разместить под телом цикла, используя специальный синтаксис do..while:

```
1  do {  
2    // тело цикла  
3  } while (condition);
```

Цикл “do...while”

Цикл сначала выполнит тело, а затем проверит условие `condition`, и пока его значение равно `true`, он будет выполняться снова и снова.

```
1 let i = 0;
2 do {
3     alert( i );
4     i++;
5 } while (i < 3);
```

Такая форма синтаксиса оправдана, если вы хотите, чтобы тело цикла выполнилось **хотя бы один раз**

Методы работы со строками

Доступ к элементам
массива по индексу

```
let text = "Hello, World!";
```

// Длина строки

```
let length = text.length; // 13
```

// Преобразование в верхний/нижний регистр

```
let upperCase = text.toUpperCase(); // "HELLO, WORLD!"
```

```
let lowerCase = text.toLowerCase(); // "hello, world!"
```

// Получение подстроки

```
let substring = text.substring(0, 5); // "Hello"
```

// Поиск подстроки

```
let indexOfWorld = text.indexOf("World"); // 7
```

Метод - split

Описание:

Метод split() разбивает строку на массив строк по заданному разделителю

Синтаксис:

```
str.split(separator, limit)
```

separator

Необязательный параметр. Указывает символы, используемые в качестве разделителя внутри строки.

limit

Необязательный параметр. Целое число, определяющее ограничение на количество найденных подстрок.

Метод - split

Возвращаемое значение:

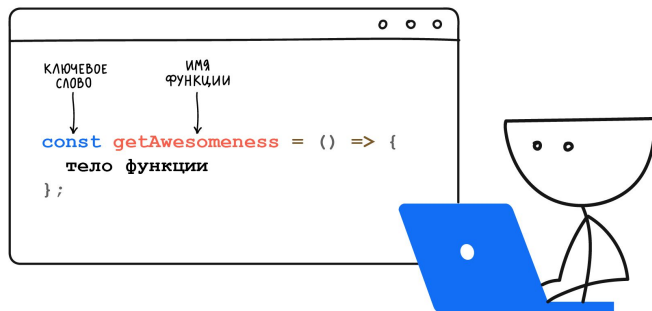
Метод split() возвращает новый массив.

Примеры:

```
1 let names = 'Вася, Петя, Маша';
2
3 let arr = names.split(', ');
4
5 for (let name of arr) {
6   alert( `Сообщение получат: ${name}.` ); // Сообщение получат: Вася (и другие имена)
7 }
```

```
1 let arr = 'Вася, Петя, Маша, Саша'.split(', ', 2);
2
3 alert(arr); // Вася, Петя
```

Функции. Основы



Зачастую нам надо повторять одно и то же действие во многих частях программы.

Например, необходимо красиво вывести сообщение при приветствии посетителя, при выходе посетителя с сайта, ещё где-нибудь.

Чтобы не повторять один и тот же код во многих местах, придуманы функции. Функции являются основными «**строительными блоками**» программы.

Объявление функции

Для создания функций мы можем использовать объявление функции.

```
1 function showMessage() {  
2   alert( 'Всем привет!' );  
3 }
```

Вначале идёт ключевое слово `function`, после него имя функции, затем список параметров в круглых скобках через запятую - такое объявление функции называется **Function Declaration**

```
1 function имя(параметры) {  
2   ...тело...  
3 }
```

Вызов функции

Наша новая функция может быть вызвана по своему имени: `showMessage()`.

```
1 function showMessage() {  
2     alert( 'Всем привет!' );  
3 }  
4  
5 showMessage();  
6 showMessage();
```

Локальные переменные

Переменные, объявленные внутри функции, видны только внутри этой функции.

```
1 function showMessage() {  
2   let message = "Привет, я JavaScript!"; // локальная переменная  
3  
4   alert( message );  
5 }  
6  
7 showMessage(); // Привет, я JavaScript!  
8  
9 alert( message ); // <-- будет ошибка, т.к. переменная видна только внутри функции
```

Внешние переменные

У функции есть доступ к внешним переменным, например:

```
1 let userName = 'Вася';  
2  
3 function showMessage() {  
4     let message = 'Привет, ' + userName;  
5     alert(message);  
6 }  
7  
8 showMessage(); // Привет, Вася
```

Если одноимённая переменная объявляется внутри функции, тогда она перекрывает внешнюю.

Параметры

Мы можем передать внутрь функции любую информацию, используя параметры. В нижеприведённом примере функции передаются два параметра: from и text.

```
1 function showMessage(from, text) { // параметры: from, text
2   alert(from + ': ' + text);
3 }
4
5 showMessage('Аня', 'Привет!'); // Аня: Привет! (*)
6 showMessage('Аня', "Как дела?"); // Аня: Как дела? (**)
```

Возврат значения

Функция может вернуть результат, который будет передан в вызвавший её код.

Простейшим примером может служить функция сложения двух чисел:

```
1 function sum(a, b) {  
2   return a + b;  
3 }  
4  
5 let result = sum(1, 2);  
6 alert( result ); // 3
```

Выбор имени функции

Функция – это действие. Поэтому имя функции обычно является глаголом. Оно должно быть кратким, точным и описывать действие функции, чтобы программист, который будет читать код, получил верное представление о том, что делает функция.

Например, функции, начинающиеся с "show" обычно что-то показывают.

Функции, начинающиеся с...

- "**get...**" – возвращают значение,

- "**calc...**" – что-то вычисляют,

- "**create...**" – что-то создают,

- "**check...**" – что-то проверяют и возвращают логическое значение, и т.д.

Выбор имени функции

Примеры имен

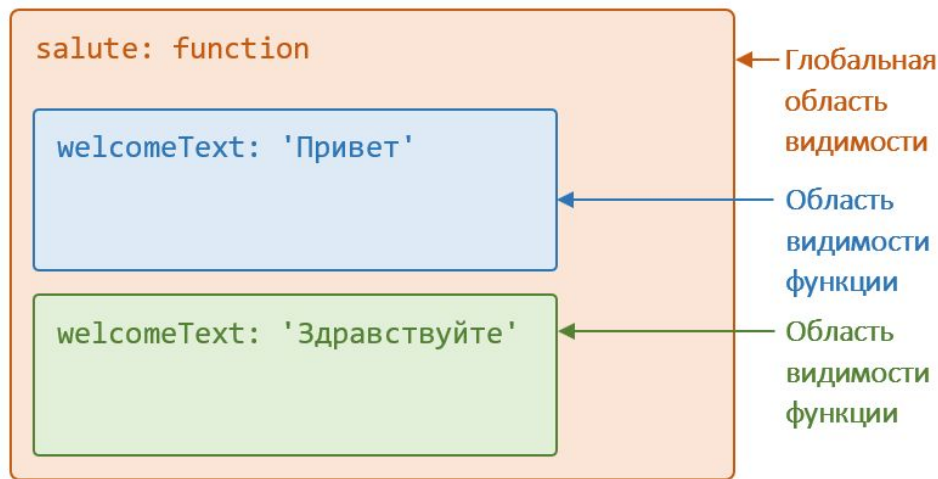
```
1 showMessage(..)    // показывает сообщение
2 getAge(..)         // возвращает возраст (получая его каким-то образом)
3 calcSum(..)        // вычисляет сумму и возвращает результат
4 createForm(..)     // создаёт форму (и обычно возвращает её)
5 checkPermission(..) // проверяет доступ, возвращая true/false
```


Области видимости переменных



Область видимости определяет, где в коде программы будут доступны переменные и функции. В JavaScript есть два типа области видимости — **глобальная** и **локальная**

Переменные `let` и `const` ведут себя аналогично, в примерах будем употреблять `let`



Блоки кода

Если переменная объявлена внутри блока кода {...}, то она видна только внутри этого блока.

```
1 {  
2   // выполняем некоторые действия с локальной переменной, которые не должны  
3  
4   let message = "Hello"; // переменная видна только в этом блоке  
5  
6   alert(message); // Hello  
7 }  
8  
9 alert(message); // ReferenceError: message is not defined
```

Для **if**, **for**, **while** и т.д. переменные, объявленные в блоке кода {...}, также видны только внутри:

```
1  for (let i = 0; i < 3; i++) {  
2    // переменная i видна только внутри for  
3    alert(i); // 0, потом 1, потом 2  
4  }  
5  
6  alert(i); // Ошибка, нет такой переменной!
```

Устаревшее ключевое слово `var`



Устаревшее ключевое слово "var".

Обычно var не используется в современных скриптах, но всё ещё может скрываться в старых.

На первый взгляд, поведение var похоже на let. Например, объявление переменной:

```
1 function sayHi() {  
2   var phrase = "Привет"; // локальная переменная, "var" вместо "let"  
3  
4   alert(phrase); // Привет  
5 }  
6  
7 sayHi();  
8  
9 alert(phrase); // Ошибка: phrase не определена
```

Для «**var**» не существует блочной области видимости

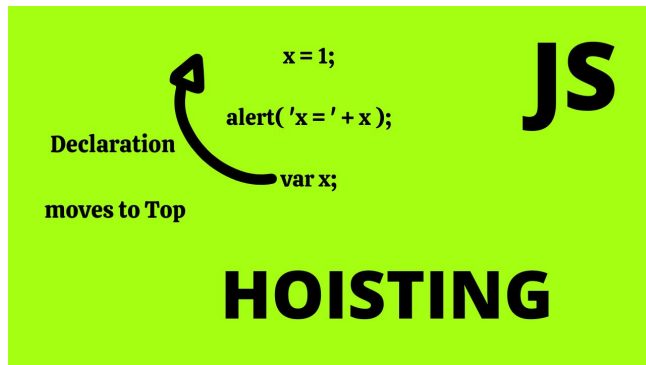
Область видимости переменных **var** ограничивается либо функцией, либо, если переменная глобальная, то скриптом. Такие переменные доступны за пределами блока (if, for, while и т.д).

```
1  if (true) {  
2    var test = true; // используем var вместо let  
3  }  
4  
5  alert(test); // true, переменная существует вне блока if
```

Так как var игнорирует блоки, мы получили глобальную переменную test.

Поднятие или **hoisting** — это механизм в JavaScript, в котором переменные и объявления функций, передвигаются вверх своей области видимости перед тем, как код будет выполнен.

Это позволяет вам использовать переменные и функции до их явного объявления в коде.





Ваша новая IT-профессия – Ваш новый уровень жизни

Программирование с нуля в
немецкой школе AIT TR GmbH