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)

Повторим;)

Какие особенности массива в JS?

Что делают методы push, shift, unshift, pop?

Как вызвать определенный элемент массива

ЦЕЛЬ

Изучить методы строк и функции в JS

ПЛАН ЗАНЯТИЯ

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

Методы строк

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:

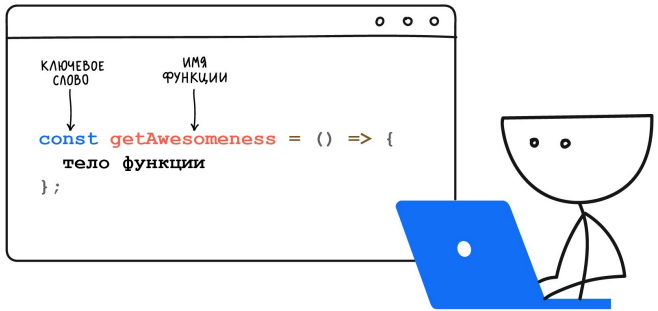
Метод split() возвращает новый массив.

Примеры:

```
let names = 'Вася, Петя, Маша';

let arr = names.split(', ');

for (let name of arr) {
  alert( `Сообщение получат: ${name}.` ); // Сообщение получат: Вася (и другие имена)
}
```

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

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

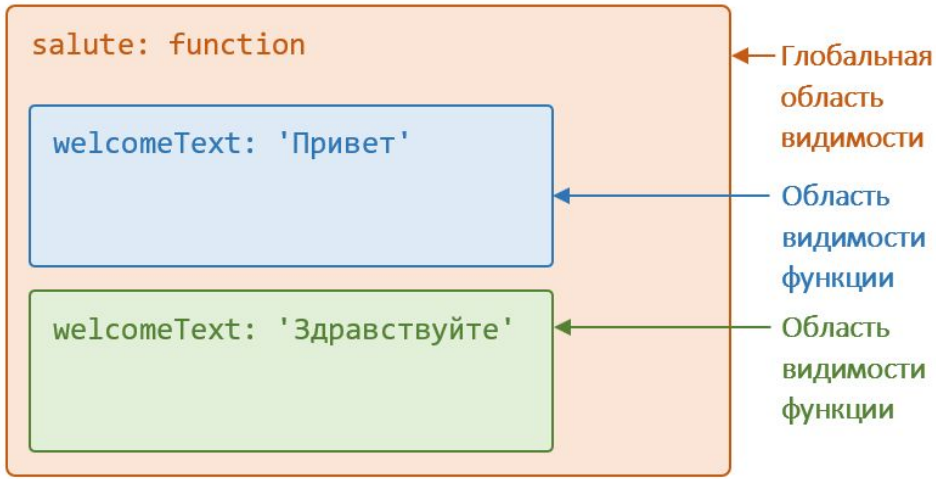
Примеры имен

```
showMessage(..)     // показывает сообщение
getAge(..)          // возвращает возраст (получая его каким-то образом)
calcSum(..)         // вычисляет сумму и возвращает результат
createForm(..)      // создаёт форму (и обычно возвращает её)
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.

```
{
  // выполняем некоторые действия с локальной переменной, которые не должны
  let message = "Hello"; // переменная видна только в этом блоке
  alert(message); // Hello
}
alert(message); // ReferenceError: message is not defined
```

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

```
for (let i = 0; i < 3; i++) {
  // переменная i видна только внутри for
  alert(i); // 0, потом 1, потом 2
}

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

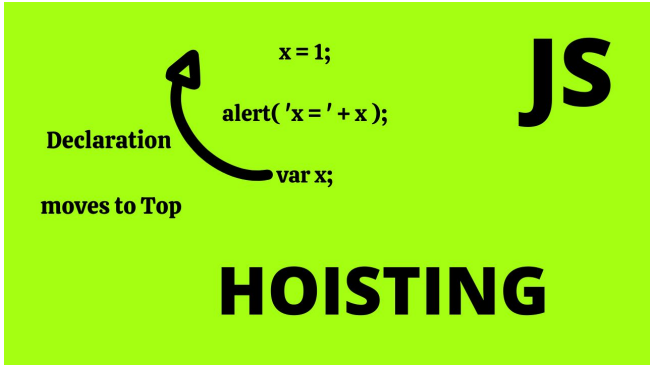
Область видимости переменных **var** ограничивается либо функцией, либо, если переменная глобальная, то скриптом. Такие переменные доступны за пределами блока (if, for, while и т.д).

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

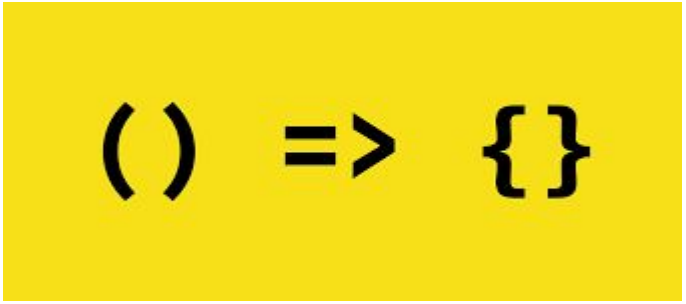
Так как var игнорирует блоки, мы получили глобальную переменную test.

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.



Function types



Существует ещё один синтаксис создания функций, который называется **Function Expression** (Функциональное Выражение).
Данный синтаксис позволяет нам создавать новую функцию в середине любого выражения.

```
let sayHi = function() {
  alert( "Привет" );
};
```

Function Expression

```
function sayHi() {
  alert( "Привет" );
}
```

Function Declaration

Разница Function Expression и Function Declaration

1. **Function Declaration** может быть вызвана раньше, чем она объявлена.

```
1  sayHi("Вася"); // Привет, Вася
2
3  function sayHi(name) {
4    alert( `Привет, ${name}` );
5  }
```

1. **Function Expression** создаётся, когда выполнение доходит до него, и затем уже может использоваться.

```
1  sayHi("Вася"); // ошибка!
2
3  let sayHi = function(name) {  // (*) магии больше нет
4    alert( `Привет, ${name}` );
5  };
```

Стрелочные функции

Существует ещё один очень простой и лаконичный синтаксис для создания функций. Он называется «функции-стрелки» или «стрелочные функции» (arrow functions), т.к. выглядит следующим образом:

```
let func = (arg1, arg2, ...argN) => expression;
```

Это создаёт функцию func, которая принимает аргументы arg1..argN, затем вычисляет expression в правой части с их использованием и возвращает результат.

Обычная функция

```
let func = function(arg1, arg2, ...argN) {
  return expression;
};
```

Стрелочные функции

Если у нас только один аргумент, то круглые скобки вокруг параметров можно опустить, сделав запись ещё короче:

```
1  let double = n => n * 2;
2  // примерно тоже что и: let double = function(n) { return n * 2 }
3
4  alert( double(3) ); // 6
```

Многострочные стрелочные функции

Иногда нам нужна более сложная функция, с несколькими выражениями и инструкциями. Это также возможно, нужно лишь заключить их в фигурные скобки. При этом важное отличие – в том, что в таких скобках для возврата значения нужно использовать return (как в обычных функциях).

```
let sum = (a, b) => {
    let result = a + b;
// если мы используем фигурные скобки, то нам нужно явно указать "return"
    return result;
};

alert( sum(1, 2) );
```

Функция в качестве параметра

Функция может передаваться в качестве аргумента при вызове другой функции. Например, функция, которая может выполнить произвольную операцию между двумя числами.

```
function operateOnNumbers(a, b, operation) {
  return operation(a, b);
}

// Функция сложения
function add(x, y) {
  return x + y;
}

// Функция вычитания
function subtract(x, y) {
  return x - y;
}

const sumResult = operateOnNumbers(5, 3, add);
console.log(sumResult);

const differenceResult = operateOnNumbers(8, 3, subtract);
console.log(differenceResult);
```

Передача анонимной функции в качестве параметра

Анонимные функции в JavaScript - это функции, которые не имеют имени и обычно определяются прямо в месте, где они используются.

```
// Функция, принимающая функцию в качестве параметра
function performOperation(x, y, operation) {
  return operation(x, y);
}

// Используем функцию performOperation с анонимной функцией-выражением
const result = performOperation(8, 3, function(a, b) {
  return a - b;
});

console.log(result); // Вывод: 5
```

Параметры по умолчанию

Если при вызове функции аргумент не был указан, то его значением становится undefined.
Если мы хотим задать параметру text значение по умолчанию, мы должны указать его после =:

```
1  function showMessage(from, text = "текст не добавлен") {
2    alert( from + ": " + text );
3  }
4
5  showMessage("Аня"); // Аня: текст не добавлен
```

Теперь, если параметр text не указан, его значением будет "текст не добавлен"

Ваша новая IT-профессия – Ваш новый уровень жизни

Программирование с нуля в
немецкой школе AIT TR GmbH

AIT
Hi-tech team