

Курс: HTML/CSS + JS С НУЛЯ

Тренер:
Илья Литвинов

Лекция 5



Содержание

- Введение
- Структура языка
- Подключение
- Синтаксис
- Переменные
- Типы данных
- Преобразование
- Операторы
- Условные операторы
- Консоль разработчика
- Логические операторы
- Циклы while/for
- Функции



Введение

Введение

4 апреля 1995 г.



Brandan Eich

LISP

Schema

LiveScript

JavaScript

Введение

4 декабря 1995 года



ECMAScript(спецификация языка)

Введение

"JS - прототипно-ориентированный, сценарный язык программирования"

JavaScript изначально создавался для того, чтобы сделать web-странички «живыми». Программы на этом языке называются скриптами. В браузере они подключаются напрямую к HTML и, как только загружается страничка - тут же выполняются.

Программы на JavaScript - обычный текст

Процесс выполнения скрипта называют «интерпретацией».

Введение

Для выполнения программ, не важно на каком языке, существуют два способа: «компиляция» и «интерпретация».

Компиляция - это когда исходный код программы, при помощи специального инструмента, другой программы, которая называется «компилятор», преобразуется в другой язык, как правило - в машинный код. Этот машинный код затем распространяется и запускается. При этом исходный код программы остаётся у разработчика.

Интерпретация - это когда исходный код программы получает другой инструмент, который называют «интерпретатор», и выполняет его «как есть». При этом распространяется именно сам исходный код (скрипт). Этот подход применяется в браузерах для JavaScript.

Современные интерпретаторы перед выполнением преобразуют JavaScript в машинный код или близко к нему, оптимизируют, а уже затем выполняют. И даже во время выполнения стараются оптимизировать. Поэтому JavaScript работает очень быстро.

Введение

Это **полноценный язык**, программы на котором можно запускать и на сервере, и даже в стиральной машинке, если в ней установлен соответствующий интерпретатор.

Современный JavaScript - это «безопасный» язык программирования общего назначения. Он не предоставляет низкоуровневых средств работы с памятью, процессором, так как изначально был ориентирован на браузеры, в которых это не требуется.

Что же касается остальных возможностей - они зависят от окружения, в котором запущен JavaScript

Возможности JS в браузере

Создавать новые HTML-теги, удалять существующие, менять стили элементов, прятать, показывать элементы и т.п.

Реагировать на действия посетителя, обрабатывать клики мыши, перемещения курсора, нажатия на клавиатуру и т.п.

Посылать запросы на сервер и загружать данные без перезагрузки страницы (эта технология называется «AJAX»).

Получать и устанавливать cookie, запрашивать данные, выводить сообщения...

Ограничения в браузере

JavaScript не может читать/записывать произвольные файлы на жесткий диск, копировать их или вызывать программы. Он не имеет прямого доступа к операционной системе.

JavaScript, работающий в одной вкладке, не может общаться с другими вкладками и окнами, за исключением случая, когда он сам открыл это окно или несколько вкладок из одного источника (одинаковый домен, порт, протокол).

Есть способы это обойти, и они раскрыты в учебнике, но они требуют специального кода на оба документа, которые находятся в разных вкладках или окнах. Без него, из соображений безопасности, залезть из одной вкладки в другую при помощи JavaScript нельзя.

Из JavaScript можно легко посылать запросы на сервер, с которого пришла страница. Запрос на другой домен тоже возможен, но менее удобен, т. к. и здесь есть ограничения безопасности.

Уникальность JS

- Полная интеграция с HTML/CSS.
- Простые вещи делаются просто.
- Поддерживается всеми браузерами

Этих трёх вещей одновременно нет больше ни в одной браузерной технологии.

Поэтому JavaScript и является самым распространённым средством создания браузерных интерфейсов.

Справочники JS

[Mozilla Developer Network](#) - содержит информацию, верную для основных браузеров. Также там присутствуют расширения только для Firefox (они помечены).

[MSDN](#) - справочник от Microsoft. Там много информации, в том числе и по JavaScript (они называют его «JScript»). Если нужно что-то, специфичное для IE - лучше лезть сразу туда.

[Safari Developer Library](#) - менее известен и используется реже, но в нём тоже можно найти ценную информацию.

Структура Javascript

Структура JS



ECMAScript

ECMAScript не является браузерным языком и в нём не определяются методы ввода и вывода информации. Это, скорее, основа для построения скриптовых языков. Спецификация ECMAScript описывает типы данных, инструкции, ключевые и зарезервированные слова, операторы, объекты, регулярные выражения, не ограничивая авторов производных языков в расширении их новыми составляющими.

Browser Object Model

Объектная модель браузера — браузер-специфичная часть языка, являющаяся прослойкой между ядром и объектной моделью документа. Основное предназначение объектной модели браузера — управление окнами браузера и обеспечение их взаимодействия. Каждое из окон браузера представляется объектом window, центральным объектом DOM. Объектная модель браузера на данный момент не стандартизирована, однако спецификация находится в разработке WHATWG и W3C.

- управление фреймами,
- поддержка задержки в исполнении кода и зацикливания с задержкой, системные диалоги,
- управление адресом открытой страницы,
- управление информацией о браузере,
- управление информацией о параметрах монитора,
- ограниченное управление историей просмотра страниц,
- поддержка работы с HTTP cookie.

Document Object Model

Объектная модель документа — интерфейс программирования приложений для HTML и XML-документов. Согласно DOM, документ (например, веб-страница) может быть представлен в виде дерева объектов, обладающих рядом свойств, которые позволяют производить с ним различные манипуляции.

- генерация и добавление узлов,
- получение узлов,
- изменение узлов,
- изменение связей между узлами,
- удаление узлов.

```
1  <!doctype html>
2  <html>
3  <head>
4      <meta charset="UTF-8">
5      <title>Untitled Document</title>
6      <link rel="stylesheet" href="styles.css">
7      <link rel="stylesheet" href="style-donate.css">
8  </head>
9  <body>
10     <div class = "animation1">
11         <span class="one"></span>
12         <span class="two"></span>
13         <span class="three"></span>
14         <span class="four"></span>
15     </div>
16 </body>
17 </html>
```

Document Object Model

1. Теги образуют узлы-элементы (element node). Естественным образом одни узлы вложены в другие. Структура дерева образована исключительно за счет них.
2. Текст внутри элементов образует текстовые узлы (text node). Текстовый узел содержит исключительно строку текста и не может иметь потомков, то есть он всегда на самом нижнем уровне.

Подключение

Подключение

Программы на языке JavaScript можно вставить в любое место HTML при помощи тега SCRIPT

```
1  <!DOCTYPE HTML>
2  <html>
3  <head>
4      <!-- Тег meta для указания кодировки -->
5      <meta charset="utf-8">
6  </head>
7  <body>
8      <p>Начало документа...</p>
9      <script>
10         alert( 'Привет, Мир!' );
11     </script>
12
13     <p>...Конец документа</p>
14
15 </body>
16 </html>
```

Подключение внешних скриптов

```
1 <script src="/path/to/script.js"></script>
```

```
1 <script src="https://cdnjs.cloudflare.com/ajax/libs/lodash.js/4.3.0/lodash.js"></script>
```

Sync/async scripts

Браузер загружает и отображает HTML постепенно. Особенно это заметно при медленном интернет-соединении: браузер не ждёт, пока страница загрузится целиком, а показывает ту часть, которую успел загрузить.

Если браузер видит тег `<script>`, то он по стандарту обязан сначала выполнить его, а потом показать оставшуюся часть страницы.

Такое поведение называют «синхронным». Как правило, оно вполне нормально, но есть важное следствие.

Если скрипт - внешний, то пока браузер не выполнит его, он не покажет часть страницы под ним.

Sync/async scripts

Атрибут async

Поддерживается всеми браузерами, кроме IE9-. Скрипт выполняется полностью асинхронно. То есть, при обнаружении `<script async src="...">` браузер не останавливает обработку страницы, а спокойно работает дальше. Когда скрипт будет загружен - он выполнится.

Атрибут defer

Поддерживается всеми браузерами, включая самые старые IE. Скрипт также выполняется асинхронно, не заставляет ждать страницу, но есть два отличия от async.

Первое - браузер гарантирует, что относительный порядок скриптов с defer будет сохранён.

Sync/async scripts

В коде (с async) первым сработает тот скрипт, который раньше загрузится:

```
1 <script src="1.js" async></script>  
2 <script src="2.js" async></script>
```

А в таком коде (с defer) первым сработает всегда 1.js, а скрипт 2.js, даже если загрузился раньше, будет его ждать.

```
1 <script src="1.js" defer></script>  
2 <script src="2.js" defer></script>
```


Синтаксис

Синтаксис

Синтаксис JavaScript и Java сделан по образцу C и C++. Отметим основные правила:

Чувствительность к регистру. Все ключевые слова пишутся в нижнем регистре. Все переменные и названия функций пишутся точно так же, как и были определены (например, переменные Str и str являются разными переменными).

Пробелы, табуляция и перевод строки. Эти символы игнорируются в JavaScript, так что можно использовать их для форматирования кода с тем, чтобы его было удобно читать.

Символ точка с запятой (;). Все операторы должны быть разделены этим символом. Если оператор завершается переводом строки, то точку с запятой можно опустить. При этом нужно следить за тем, чтобы при разрыве строки одного оператора, новая строка не начиналась бы с самостоятельного оператора.

Комментарии. JavaScript игнорирует любой текст расположенный между символами /* и */. Также игнорируется текст начинающийся символами // и заканчивающийся концом строки.

Идентификаторы. Идентификаторами являются имена переменных, функций, а также меток. Идентификаторы образуются из любого количества букв ASCII, подчеркивания (_) и символа доллара (\$). Первым символом не может быть цифра.

Ключевые слова. Ключевые слова не могут использоваться в качестве идентификаторов. Ключевыми словами являются: break, case, continue, default, delete, do, else, export, false, for, function, if, import, in, new, null, return, switch, this, true, typeof, with.

Команды

С помощью команд вы говорите интерпретатору что он должен делать

```
1 alert('Привет');
```

Для того, чтобы добавить в код ещё одну команду - можно поставить её после точки с запятой.

```
1 alert('Привет'); alert('Мир');
```

Как правило, каждая команда пишется на отдельной строке - так код лучше читается.

Комментарии

Комментарии могут находиться в любом месте программы и никак не влияют на её выполнение. Интерпретатор JavaScript попросту игнорирует их.

```
1 // Команда ниже говорит "Привет"
2 alert( 'Привет' );
3
4 alert( 'Мир' ); // Второе сообщение выводим отдельно
```

Многострочные комментарии начинаются слешем-звездочкой «/*» и заканчиваются звездочкой-слешем «*/»

```
1 /* Пример с двумя сообщениями.
2 Это – многострочный комментарий.
3 */
4 alert( 'Привет' );
5 alert( 'Мир' );
```

Переменные

Переменные

Переменная состоит из имени и выделенной области памяти, которая ему соответствует.

Для объявления или, другими словами, создания переменной используется ключевое слово `var`:

```
1 var message;
```

После объявления, можно записать в переменную данные:

```
1 var message;  
2 message = 'Hello'; // сохраним в переменной строку
```

```
1 var message = 'Hello!';
```

Константы

Константа - это переменная, которая никогда не меняется. Как правило, их называют большими буквами, через подчёркивание. Например:

```
1 var COLOR_RED = "#F00";  
2 var COLOR_GREEN = "#0F0";  
3 var COLOR_BLUE = "#00F";  
4 var COLOR_ORANGE = "#FF7F00";  
5  
6 var color = COLOR_ORANGE;  
7 alert( color ); // #FF7F00
```

Технически, константа является обычной переменной, то есть её можно изменить. Но мы договариваемся этого не делать.

Константы используют вместо строк и цифр, чтобы сделать программу понятнее и избежать ошибок.

Имена переменных

Правильный выбор имени переменной - одна из самых важных и сложных вещей в программировании

- Никакого транслита. Только английский.
- Использовать короткие имена только для переменных «местного значения».
- Переменные из нескольких слов пишутся в camel case - вместеВотТак.
- Имя переменной должно максимально чётко соответствовать хранимым в ней данным.

Типы данных

Типы данных

В JavaScript существует 6 основных типов данных.

- number
- string
- boolean
- null
- undefined
- object

Number

```
1 var n = 123;  
2 n = 12.345;
```

Единый тип число используется как для целых, так и для дробных чисел.

Существуют специальные числовые значения Infinity (бесконечность) и NaN (ошибка вычислений).

Например, бесконечность Infinity получается при делении на ноль:

```
1 alert( 1 / 0 ); // Infinity
```

Ошибка вычислений NaN будет результатом некорректной математической операции, например:

```
1 alert( "нечисло" * 2 ); // NaN, ошибка
```

String/строка

```
1 var str = "Мама мыла раму";  
2 str = 'Одинарные кавычки тоже подойдут';
```

В JavaScript любые текстовые данные являются строками. Не существует отдельного типа «символ», который есть в ряде других языков.

Boolean

Всего два значения: **true** (истина) и **false** (ложь).

```
1 var checked = true;  
2 checked = false;
```

null

Значение `null` не относится ни к одному из типов выше, а образует свой отдельный тип, состоящий из единственного значения `null`:

```
1 var age = null;
```

В JavaScript `null` не является «ссылкой на несуществующий объект» или «нулевым указателем», как в некоторых других языках. Это просто специальное значение, которое имеет смысл «ничего» или «значение неизвестно».

Undefined

Значение `undefined`, как и `null`, образует свой собственный тип, состоящий из одного этого значения. Оно имеет смысл «значение не присвоено».

Если переменная объявлена, но в неё ничего не записано, то её значение как раз и есть `undefined`:

```
1 var x;  
2 alert( x ); // выведет "undefined"
```

Object

Первые 5 типов называют «примитивными».

Особняком стоит шестой тип: «объекты».

Он используется для коллекций данных и для объявления более сложных сущностей.

Объявляются объекты при помощи фигурных скобок {...}, например:

```
1 var user = { name: "Вася" };
```


Оператор typeof

```
1  typeof undefined // "undefined"
2
3  typeof 0 // "number"
4
5  typeof true // "boolean"
6
7  typeof "foo" // "string"
8
9  typeof {} // "object"
10
11  typeof null // "object" (1)
12
13  typeof function(){} // "function" (2)
```

Преобразование типов

Приведение типов

Всего есть три преобразования:

- Строковое преобразование.
- Числовое преобразование.
- Преобразование к логическому значению.

Строковое преобразование

Строковое преобразование происходит, когда требуется представление чего-либо в виде строки.

Явное приведении:

```
var strNumber = String(number); //"20"
```

Не явное:

```
1 var number = 20,  
2   strNumber = "" + number;
```

Численное преобразование

Численное преобразование происходит в математических функциях и выражениях, а также при сравнении данных различных типов (кроме сравнений `===`, `!==`).

Для преобразования к числу в явном виде можно вызвать `Number(val)`, либо, что короче, поставить перед выражением унарный плюс `+`:

```
1 var a = +"123"; // 123
2 var a = Number("123");
```

Численное преобразование

Значение	Преобразуется в...
undefined	NaN
null	0
true / false	1 / 0
Строка	Пробельные символы по краям обрезаются. Далее, если остаётся пустая строка, то 0, иначе из непустой строки "считывается" число, при ошибке результат NaN.

Логическое преобразование

Преобразование к true/false происходит в логическом контексте, таком как `if(value)`, и при применении логических операторов.

<code>undefined, null</code>	<code>false</code>
Числа	Все true, кроме 0, NaN -- false.
Строки	Все true, кроме пустой строки "" -- false
Объекты	Всегда true

Операторы

Типы операторов

Для работы с переменными, со значениями, JavaScript поддерживает все стандартные операторы, большинство которых есть и в других языках программирования.

Операнд - то, к чему применяется оператор. Например: $5 * 2$ - оператор умножения с левым и правым операндами. Другое название: «аргумент оператора».

Унарным называется оператор, который применяется к одному выражению. Например, оператор унарный минус "-" меняет знак числа на противоположный.

Бинарным называется оператор, который применяется к двум операндам. Тот же минус существует и в бинарной форме.

Оператор присваивание

Присваивание	$x = y$	$x = y$
Присваивание со сложением	$x += y$	$x = x + y$
Присваивание с вычитанием	$x -= y$	$x = x - y$
Присваивание с умножением	$x *= y$	$x = x * y$
Присваивание с делением	$x /= y$	$x = x / y$

Операторы сравнения

Равно (==)	Возвращает true, если операнды равны.	<pre>3 == var1 "3" == var1 3 == '3'</pre>
Не равно (!=)	Возвращает true, если операнды не равны.	<pre>var1 != 4 var2 != "3"</pre>
===)	Возвращает true, если операнды равны и имеют одинаковый тип. См. также Object.is и sameness in JS .	<pre>3 === var1</pre>
Строго не равно (!==)	Возвращает true, если операнды не равны и/или имеют разный тип.	<pre>var1 !== "3" 3 !== '3'</pre>


Операторы сравнения

Больше (>)	Возвращает true, если операнд слева больше операнда справа.	<code>var2 > var1</code> <code>"12" > 2</code>
Больше или равно (>=)	Возвращает true, если операнд слева больше или равен операнду справа.	<code>var2 >= var1</code> <code>var1 >= 3</code>
Меньше (<)	Возвращает true, если операнд слева меньше операнда справа.	<code>var1 < var2</code> <code>"2" < "12"</code>
Меньше или равно (<=)	Возвращает true, если операнд слева меньше или равен операнду справа.	<code>var1 <= var2</code> <code>var2 <= 5</code>

Арифметические операторы

Остаток от деления (%)	Бинарный оператор. Возвращает целочисленный остаток от деления двух операндов.	12 % 5 вернёт 2.
Инкремент (++)	Унарный оператор. Добавляет единицу к своему операнду. Если используется в качестве префикса (++x), то возвращает значение операнда с добавленной к нему единицей; а в случае применения в качестве окончания (x++) возвращает величину операнда перед добавлением к нему единицы.	Если x равно 3, тогда ++x установит значение x равным 4 и вернёт 4, напротив x++ вернёт 3 и потом установит значение x равным 4.
Декремент (--)	Унарный оператор. Вычитает единицу из своего операнда. Логика данного оператора аналогична оператору инкремента.	Если x равно 3, тогда --x установит значение x равным 2 и вернёт 2, напротив x-- вернёт 3 и потом установит значение x равным 2.

Арифметические операторы

Унарный минус–	Унарный оператор. Возвращает отрицательное значение своего операнда.	Если x равно 3, тогда –x вернёт -3.
Унарный плюс (+)	Унарный оператор. Пытается конвертировать операнд в число, если он ещё не оно.	+"3" вернёт 3. +true вернёт 1.
Возведение в степень (**) 	Возводит основание в показатель степени, как, основание ^{степень}	2 ** 3 вернёт 8. 10 ** -1 вернёт 0.1.

Логические операторы

<code>&&</code>	<code>expr1 && expr2</code>	(Логическое И) Возвращает операнд <code>expr1</code> если он может быть преобразован в <code>false</code> ; в противном случае возвращает операнд <code>expr2</code> . Таким образом, при использовании булевых величин в качестве операндов, оператор <code>&&</code> возвращает <code>true</code> если оба операнда <code>true</code> ; в противном случае возвращает <code>false</code> .
<code> </code>	<code>expr1 expr2</code>	(Логическое ИЛИ) Возвращает операнд <code>expr1</code> если он может быть преобразован в <code>true</code> ; в противном случае возвращает операнд <code>expr2</code> . Таким образом, при использовании булевых величин в качестве операндов, оператор <code> </code> возвращает <code>true</code> если один из операндов <code>true</code> ; если же оба <code>false</code> , то возвращает <code>false</code> .
<code>!</code>	<code>!expr</code>	(Логическое НЕ) Возвращает <code>false</code> если операнд может быть преобразован в <code>true</code> ; в противном случае возвращает <code>true</code> .

Условные операторы

Условный оператор

Оператор if («если») получает условие, в примере выше это `year != 2011`. Он вычисляет его, и если результат - `true`, то выполняет команду.

```
1 if (year != 2011) {  
2   alert( 'А вот..' );  
3   alert( '..и неправильно!' );  
4 }
```

Рекомендуется использовать фигурные скобки всегда, даже когда команда одна.

Преобразование к логическому типу

Оператор `if (...)` вычисляет и преобразует выражение в скобках к логическому типу.

В логическом контексте:

Число `0`, пустая строка `""`, `null` и `undefined`, а также `NaN` являются `false`,
Остальные значения - `true`.

else/неверное условие

Необязательный блок else («иначе») выполняется, если условие неверно:

```
1 var year = prompt('Введите год появления стандарта ECMA-262 5.1', '');
2
3 if (year == 2011) {
4     alert( 'Да вы знаток!' );
5 } else {
6     alert( 'А вот и неправильно!' ); // любое значение, кроме 2011
7 }
```

Несколько условий else if

Бывает нужно проверить несколько вариантов условия. Для этого используется блок else if

```
1 var year = prompt('В каком году появилась спецификация ECMA-262 5.1?', '');
2
3 if (year < 2011) {
4     alert( 'Это слишком рано..' );
5 } else if (year > 2011) {
6     alert( 'Это поздновато..' );
7 } else {
8     alert( 'Да, точно в этом году!' );
9 }
```

Тернарный оператор ? :

Иногда нужно в зависимости от условия присвоить переменную

```
1 условие ? значение1 : значение2
```

```
1 access = (age > 14) ? true : false;
```

Тернарный оператор ? :

```
1 var age = prompt('возраст?', 18);
2
3 var message = (age < 3) ? 'Здравствуй, малыш!' :
4   (age < 18) ? 'Привет!' :
5   (age < 100) ? 'Здравствуйте!' :
6   'Какой необычный возраст!';
7
8 alert( message );
```

```
1 if (age < 3) {
2   message = 'Здравствуй, малыш!';
3 } else if (age < 18) {
4   message = 'Привет!';
5 } else if (age < 100) {
6   message = 'Здравствуйте!';
7 } else {
8   message = 'Какой необычный возраст!';
9 }
```

Логические операторы

Логические операторы

Для операций над логическими значениями в JavaScript есть `||` (ИЛИ), `&&` (И) и `!` (НЕ).

Хоть они и называются «логическими», но в JavaScript могут применяться к значениям любого типа и возвращают также значения любого типа.

|| или

Оператор ИЛИ выглядит как двойной символ вертикальной черты:

```
1 result = a || b;
```

```
alert( true || true ); // true  
alert( false || true ); // true  
alert( true || false ); // true  
alert( false || false ); // false
```

Если значение не логического типа - то оно к нему приводится в целях вычислений. Например, число 1 будет воспринято как true, а 0 - как false:

```
alert(1 || 0); // 1
```

|| запинаяется на «правде».

&& и

Оператор И пишется как два амперсанда &&:

```
1 result = a && b;
```

```
1 alert( true && true ); // true
2 alert( false && true ); // false
3 alert( true && false ); // false
4 alert( false && false ); // false
```

```
1 var hour = 12,
2     minute = 30;
3
4 if (hour == 12 && minute == 30) {
5     alert( 'Время 12:30' );
6 }
```

! не

Оператор НЕ - самый простой. Он получает один аргумент

```
alert( !true ); // false  
alert( !0 ); // true
```

Действия !:

Сначала приводит аргумент к логическому типу true/false.
Затем возвращает противоположное значение.

Циклы

Циклы

При написании скриптов зачастую встает задача сделать однотипное действие много раз.

Например, вывести товары из списка один за другим. Или просто перебрать все числа от 1 до 10 и для каждого выполнить одинаковый код.

Для многократного повторения одного участка кода - предусмотрены циклы.

While

```
while (условие) {  
    // код, тело цикла  
}
```

```
var i = 0;  
while (i < 3) {  
    alert( i );  
    i++;  
}
```

Повторение цикла по-научному называется «итерация». Цикл в примере выше совершает три итерации.

Условие в скобках интерпретируется как логическое значение, поэтому вместо `while (i!=0)` обычно пишут `while (i)`:

```
var i = 3;  
while (i) {  
    alert( i );  
    i--;  
}
```

do...while

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

```
do {  
    // тело цикла  
} while (условие);
```

Цикл, описанный, таким образом, сначала выполняет тело, а затем проверяет условие.

```
var i = 0;  
do {  
    alert( i );  
    i++;  
} while (i < 3);
```

for

Чаще всего применяется цикл for. Выглядит он так:

```
for (начало; условие; шаг) {  
    // ... тело цикла ...  
}
```

```
var i;
```

```
for (i = 0; i < 3; i++) {  
    alert( i );  
}
```


break

Выйти из цикла можно не только при проверке условия но и, вообще, в любой момент. Эту возможность обеспечивает директива `break`. Например, следующий код подсчитывает сумму вводимых чисел до тех пор, пока посетитель их вводит, а затем - выдаёт:

```
var sum = 0;

while (true) {

    var value = +prompt("Введите число", "");

    if (!value) break; // (*)

    sum += value;

}
alert( 'Сумма: ' + sum );
```

continue

Директива `continue` прекращает выполнение текущей итерации цикла. Она - в некотором роде «младшая сестра» директивы `break`: прерывает не весь цикл, а только текущее выполнение его тела, как будто оно закончилось. Её используют, если понятно, что на текущем повторе цикла делать больше нечего.

Например, цикл ниже использует `continue`, чтобы не выводить чётные значения:

```
for (var i = 0; i < 10; i++) {  
    if (i % 2 == 0) continue;  
    alert(i);  
}
```

Методы и свойства

Методы и свойства

Все значения в JavaScript, за исключением null и undefined, содержат набор вспомогательных функций и значений, доступных «через точку».

Такие функции называют «методами», а значения - «свойствами».

object.value

```
1 var str = "Привет, мир!";  
2 alert( str.length ); // 12
```

```
1 var n = 12.345;  
2  
3 alert( n.toFixed(2) ); // "12.35"  
4 alert( n.toFixed(0) ); // "12"  
5 alert( n.toFixed(5) ); // "12.34500"
```



Числа

- parseInt()/parseFloat()
- isNaN()
- n.toString()
- Math.floor()/Math.ceil()/Math.round()
- n.toFixed()



Строки

- .length;
- .toLowerCase();
- .toUpperCase();
- .indexOf() lastIndexOf();
- .slice();
- .charAt()/str[number] charCodeAt();
- .search(regExp);
- .match(regExp);
- .replace(regExp);
- https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/String



Функции

Функции

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

Например, красиво вывести сообщение необходимо 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.

Примеры встроенных функций вы уже видели - это `alert(message)`, `prompt(message, default)` и `confirm(question)`. Но можно создавать и свои.

```
1 function showMessage() {  
2     alert( 'Привет всем присутствующим!' );  
3 }
```


Параметры

При вызове функции ей можно передать данные, которые та использует по своему усмотрению.

```
1 function showMessage(from, text) { // параметры from, text
2
3     from = "** " + from + " **"; // здесь может быть сложный код оформления
4
5     alert(from + ': ' + text);
6 }
7
8 showMessage('Маша', 'Привет!');
9 showMessage('Маша', 'Как дела?');
```



Возврат значений

Функция может вернуть результат, который будет передан в вызвавший её код. Например, создадим функцию `calcD`, которая будет возвращать дискриминант квадратного уравнения по формуле $b^2 - 4ac$:

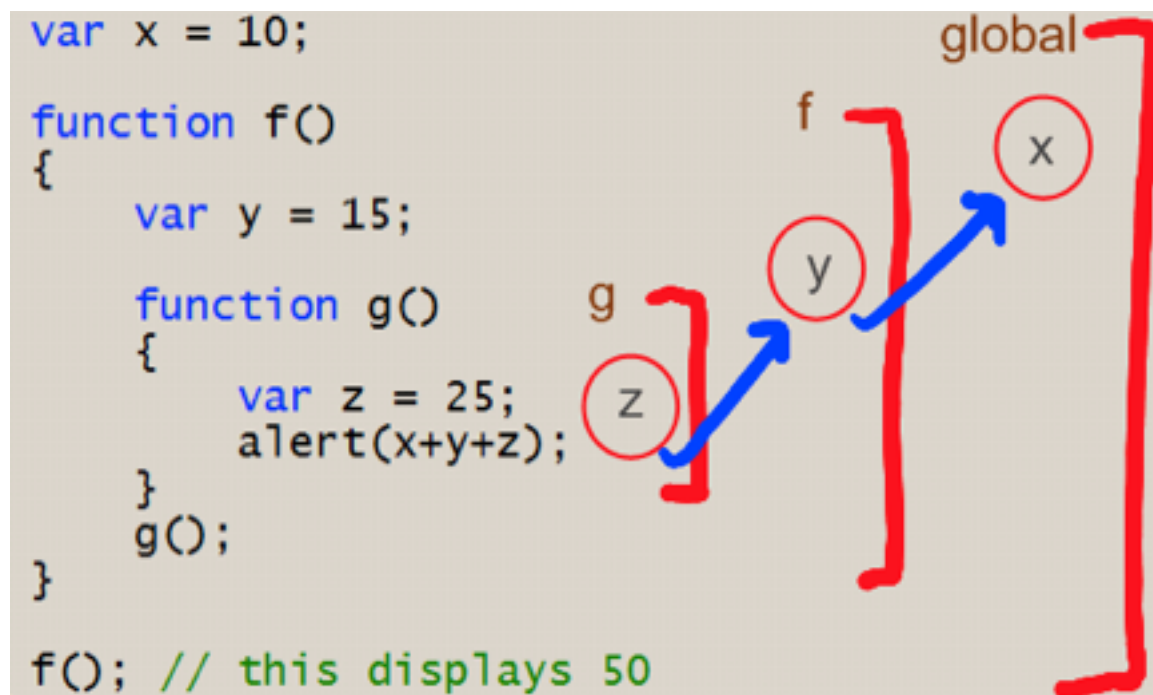
```
1 function calcD(a, b, c) {  
2   return b*b - 4*a*c;  
3 }  
4  
5 var test = calcD(-4, 2, 1);  
6 alert(test); // 20
```

Для возврата значения используется директива `return`.

Она может находиться в любом месте функции. Как только до неё доходит управление - функция завершается и значение передается обратно.

Локальная/глобальная область видимости

Функция - единственная конструкция в JS которая образует локальную область видимости. То есть переменные объявленные внутри функции не будут доступны наружу



Переменные, объявленные на уровне всего скрипта, называют «глобальными переменными».

Лексическое окружение

Все переменные внутри функции - это свойства специального внутреннего объекта `LexicalEnvironment`, который создаётся при её запуске.

Мы будем называть этот объект «лексическое окружение» или просто «объект переменных».

В отличие от `window`, объект `LexicalEnvironment` является внутренним, он скрыт от прямого доступа.

```
1 function sayHi(name) {  
2   // LexicalEnvironment = { name: 'Вася', phrase: undefined }  
3   var phrase = "Привет, " + name;  
4   alert( phrase );  
5 }  
6  
7 sayHi( 'Вася' );
```

Поднятие переменных

Что выведет следующий код?

```
var foo = 1;  
function bar() {  
    if (!foo) {  
        var foo = 10;  
    }  
    alert(foo);  
}  
bar();
```

Поднятие переменных

«В языке JavaScript допускается вставлять несколько инструкций `var` в функции, и все они будут действовать, как если бы объявления переменных находились в начале функции. Такое поведение называется *подъемом* (*hoisting*) и может приводить к логическим ошибкам, когда переменная сначала используется, а потом объявляется ниже в этой же функции.»

В языке JavaScript переменная считается объявленной, если она используется в той же области видимости (в той же функции), где находится объявление `var`, даже если это объявление располагается ниже.

```
myname = "global";  
function func() {  
    console.log(myname);  
    var myname = "local";  
    console.log(myname);  
}  
func();
```

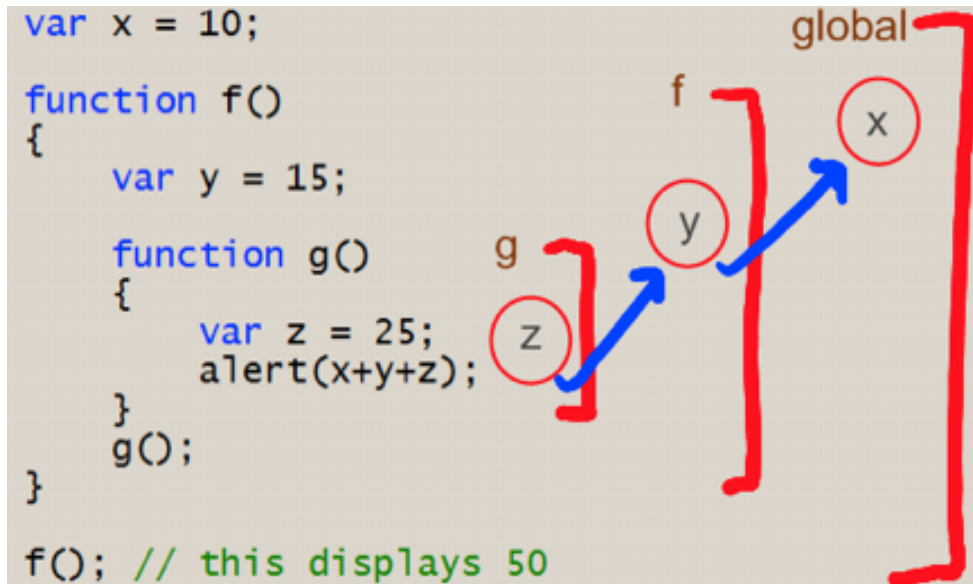
Поднятие переменных

```
myname = "global";  
function func() {  
    ↑ console.log(myname);  
    var myname = "local";  
    console.log(myname);  
}  
func();
```

```
myname = "global"; // глобальная переменная  
function func() {  
    var myname;  
  
    alert(myname); // "undefined"  
    myname = "local";  
    alert(myname); // "local"  
}  
func();
```

Замыкания

Замыкание - это функция вместе со всеми внешними переменными, которые ей доступны.



Обычно, говоря «замыкание функции», подразумевают не саму эту функцию, а именно внешние переменные.

Сохранение временной в замыкании

```
var counter = counterCreator();  
  
counter(); // => 1;  
counter(); // => 2  
function counterCreator () {  
    var totalCount = 0;  
    return function () {  
        totalCount++;  
        console.log(totalCount);  
    }  
}
```

переменная
в замыкании

Функции

функция - это всего лишь разновидность значения переменной.

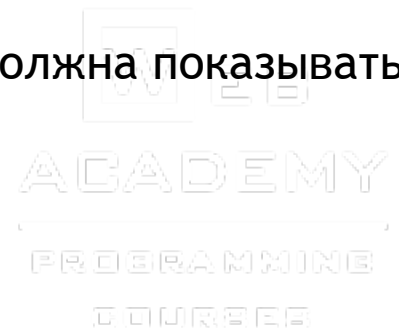
Одна функция - одно действие

Функция должна делать только то, что явно подразумевается её названием. И это должно быть одно действие.

Если оно сложное и подразумевает поддействия - может быть имеет смысл выделить их в отдельные функции? Зачастую это имеет смысл, чтобы лучше структурировать код.

...Но самое главное - в функции не должно быть ничего, кроме самого действия и поддействий, неразрывно связанных с ним.

Например, функция проверки данных (скажем, "validate") не должна показывать сообщение об ошибке. Её действие - проверить.



Функциональное выражение

Существует альтернативный синтаксис для объявления функции, который ещё более наглядно показывает, что функция - это всего лишь разновидность значения переменной.

```
1 var f = function(параметры) {  
2     // тело функции  
3 };
```

«Классическое» объявление функции, о котором мы говорили до этого, вида `function имя(параметры) {...}`, называется в спецификации языка «[Function Declaration](#)».

[Function Declaration](#) - функция, объявленная в основном потоке кода.

[Function Expression](#) - объявление функции в контексте какого-либо выражения, например присваивания.

Функциональное выражение

```
1 // Function Declaration
2 function sum(a, b) {
3     return a + b;
4 }
5
6 // Function Expression
7 var sum = function(a, b) {
8     return a + b;
9 }
```

Конец