

W



Web Academy
Programming Courses

Front-end с нуля

Тренер: Зинченко Андрей

Лекция 5

web-academy.com.ua

A

Содержание

- 1. Введение
- 2.Структура
- 3.Подключение
- 4.Синтаксис
- 5.Переменные
- 6.Типы данных
- 7.Преобразование
- 8.Операторы
- 9.Условные операторы
- 10.Консоль разработчика
- 11.Логические операторы
- 12.Циклы while/for
- 13.Функции

web-academy.com.ua

W



A



Введение

"JS - прототипно-ориентированный, сценарный язык программирования"



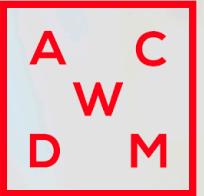
JavaScript изначально создавался для того, чтобы сделать web-страницы «живыми». Программы на этом языке называются скриптами. В браузере они подключаются напрямую к HTML и, как только загружается страница – тут же выполняются.

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».

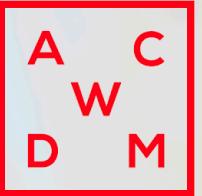
Компиляция – это когда исходный код программы, при помощи специального инструмента, другой программы, которая называется «компилятор», преобразуется в другой язык, как правило – в машинный код. Этот машинный код затем распространяется и запускается. При этом исходный код программы остаётся у разработчика.

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



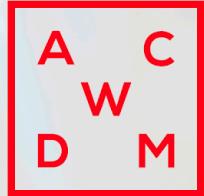
Это **полноценный язык**, программы на котором можно запускать и на сервере, и даже в стиральной машинке, если в ней установлен соответствующий интерпретатор.

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...

A



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.

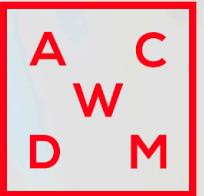
JavaScript, работающий в одной вкладке, не может общаться с другими вкладками и окнами, за исключением случая, когда он сам открыл это окно или несколько вкладок из одного источника (одинаковый домен, порт, протокол).

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

W

Структура JS



JavaScript

ECMAScript

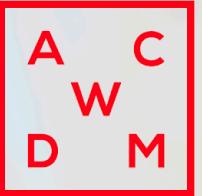
DOM

BOM

A



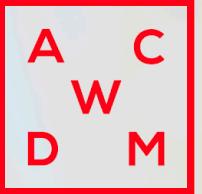
ECMAScript



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

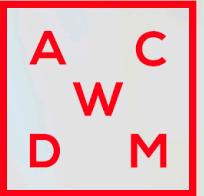


Объектная модель браузера — браузер-специфичная часть языка, являющаяся прослойкой между ядром и объектной моделью документа. Основное предназначение объектной модели браузера — управление окнами браузера и обеспечение их взаимодействия. Каждое из окон браузера представляется объектом `window`, центральным 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

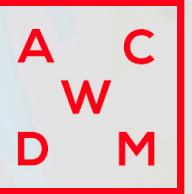


Объектная модель документа — интерфейс программирования приложений для HTML и XML-документов. Согласно DOM, документ (например, веб-страница) может быть представлен в виде дерева объектов, обладающих рядом свойств, которые позволяют производить с ним различные манипуляции.

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



Document Object Model

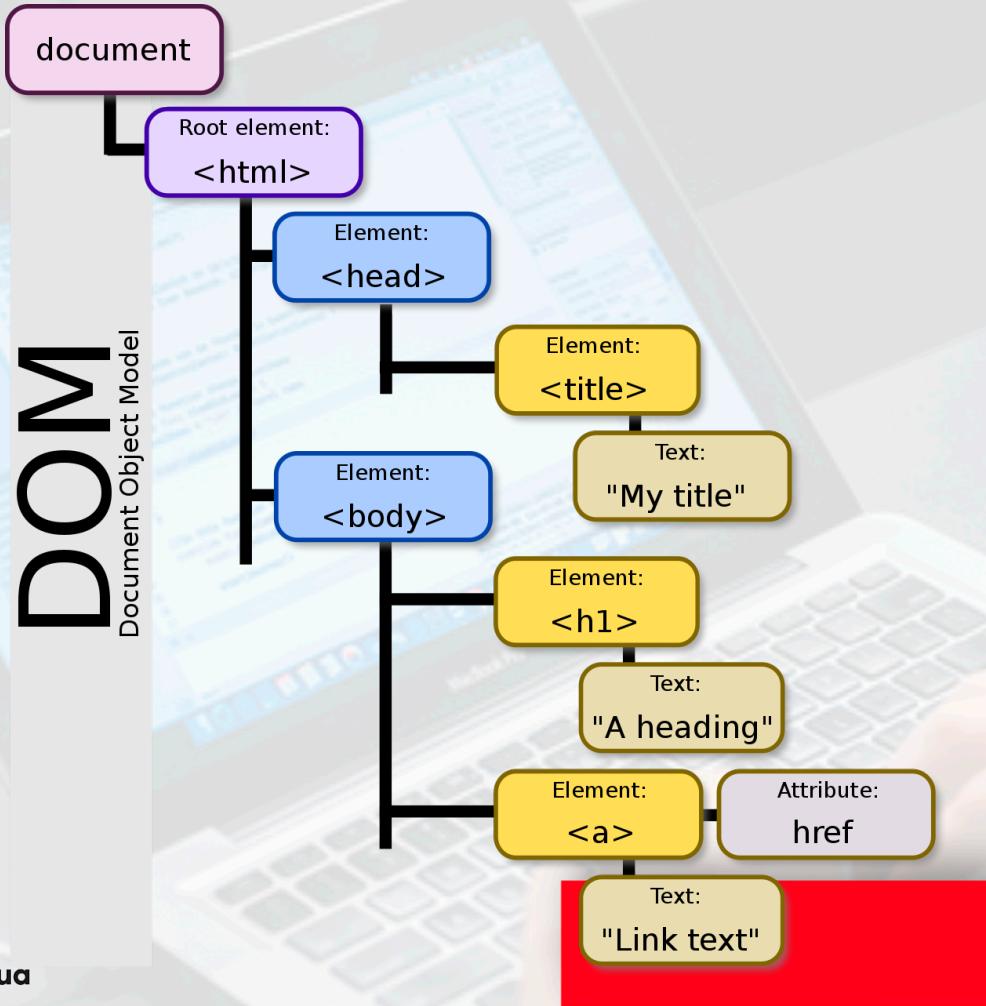


1. Теги образуют узлы-элементы (element node). Естественным образом одни узлы вложены в другие. Структура дерева образована исключительно за счет них.
2. Текст внутри элементов образует текстовые узлы (text node). Текстовый узел содержит исключительно строку текста и не может иметь потомков, то есть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

W

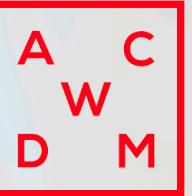
DOM

Document Object Model

**A****A C
W D M**



Подключение



A

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
    <link rel="stylesheet" href="style.css">
    <script src="index.js"></script>
</head>
<body>
    <script>
        alert('script2')
    </script>
    <div>test</div>
</body>
</html>
```



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.

A



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.

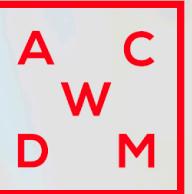
Атрибут **defer**

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.

W

sync/async scripts



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

```
<script defer src="3.js"></script>
<script defer src="4.js"></script>
```

Синтаксис



Синтаксис JavaScript и Java сделан по образцу С и С++. 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.



Синтаксис



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

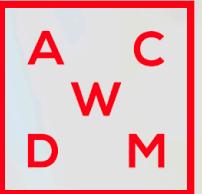
Идентификаторы. Идентификаторами являются имена переменных, функций, а также меток. Идентификаторы образуются из любого количества букв ASCII, подчеркивания (_) и символа доллара (\$). Первым символом не может быть цифра.

Ключевые слова. Ключевые слова не могут использоваться в качестве индентификаторов. 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. Для того, чтобы добавить в код ещё одну команду – можно поставить её после точки с запятой. Как правило, каждая команда пишется на отдельной строке – так код лучше читается.

```
<script>
    alert('Hello');
    alert('Hello'); alert("World")
</script>
```



Переменные

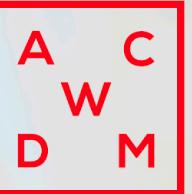


Переменная – это «именованное хранилище» для данных. Мы можем использовать переменные для хранения товаров, посетителей и других данных.

```
1 let message;  
2 message = 'Hello!';  
3  
4 alert(message); // показывает содержимое переменной
```



Переменные



```
1 let message = 'Hello!'; // определяем переменную и присваиваем ей значение  
2  
3 alert(message); // Hello!
```

```
1 let user = 'John', age = 25, message = 'Hello';
```

Переменные



Многострочный вариант немного длиннее, но легче для чтения:

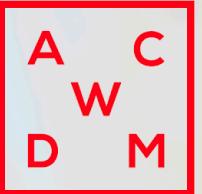
```
1 let user = 'John';
2 let age = 25;
3 let message = 'Hello';
```

```
1 let user = 'John',
2     age = 25,
3     message = 'Hello';
```

```
1 let user = 'John'
2     , age = 25
3     ; message = 'Hello';
```



Константы



Чтобы объявить константную, то есть, неизменяемую переменную, используйте `const` вместо `let`. Переменные, объявленные с помощью `const`, называются «константами». Их нельзя изменить. Попытка сделать это приведёт к ошибке. Если программист уверен, что переменная никогда не будет меняться, он может гарантировать это и наглядно донести до каждого, объявив её через `const`.

```
1 const myBirthday = '18.04.1982';
2
3 myBirthday = '01.01.2001'; // ошибка, константу нельзя перезаписать!
```



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

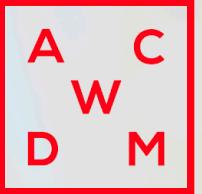


В JavaScript существует 6 основных типов данных.

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

W

Number



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:

W

String

**A C
W M
D M**

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

```
1 let str = "Привет";
2 let str2 = 'Одинарные кавычки тоже подойдут';
3 let phrase = `Обратные кавычки позволяют встраивать переменные ${str}`;
```

Boolean



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

```
1 let nameFieldChecked = true; // да, поле отмечено  
2 let ageFieldChecked = false; // нет, поле не отмечено
```

W

null



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



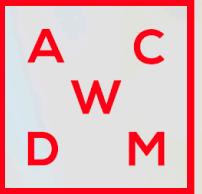
Значение `undefined`, как и `null`, образует свой собственный тип, состоящий из одного этого значения. Оно имеет смысл «значение не присвоено».

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

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

W

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:

A

W

Object

A	C
W	
D	M

```
const obj = { param: "some param" };

const user = {
    info: {
        name: "some name",
        secondName: "some second name"
    },
    phone: 11122233444,
};
```

A

typeof

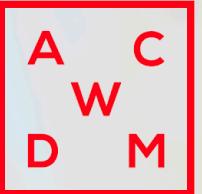
A	C
W	
D	M

```
1  typeof undefined // "undefined"
2
3  typeof 0 // "number"
4
5  typeof true // "boolean"
6
7  typeof "foo" // "string"
8
9  typeof Symbol("id") // "symbol"
10
11 typeof Math // "object"  (1)
12
13 typeof null // "object"  (2)
14
15 typeof alert // "function"  (3)
```

A



Преобразование типов



Всего есть три преобразования:

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

W Строковое преобразование



```
let str1 = 12345;  
let str2 = 67890;  
  
alert(String(str1));  
alert(" " + str2);
```

Числовое преобразование



```
let str1 = "12345";
let str2 = "67890";

alert(Number(str1));
alert(+str2);|
```

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

W Логическое преобразование



Boolean(`value`)

undefined, null

false

Числа

Все true, кроме 0, NaN -- false.

Строки

Все true, кроме пустой строки "" -- false

Объекты

Всегда true

Операторы



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 operandами. 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 operandам. Тот же минус существует и в бинарной форме.

Операторы



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 operandами. 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 operandам.



Оператор присвоения

A C
W M
D

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

$x /= y$

$x = x / y$

Оператор сравнения

A C
W M

Равно (==)

Возвращает true, если операнды равны.

```
3 == var1
"3" == var1
```

```
3 == '3'
```

Не равно (!=)

Возвращает true, если операнды не равны.

```
var1 != 4
var2 != "3"
```

===(

Возвращает true, если операнды равны и имеют одинаковый тип. См. также [Object.is](#) и [sameness in JS](#).

```
3 === var1
```

Строго не равно (!==)

Возвращает true, если операнды не равны и/или имеют разный тип.

```
var1 !== "3"
3 !== '3'
```

Оператор сравнения

A	C
W	
D	M

Больше (>)	Возвращает true, если operand слева больше operand справа.	var2 > var1 "12" > 2
Больше или равно (>=)	Возвращает true, если operand слева больше или равен operandу справа.	var2 >= var1 var1 >= 3
Меньше (<)	Возвращает true, если operand слева меньше operandа справа.	var1 < var2 "2" < "12"
Меньше или равно (<=)	Возвращает true, если operand слева меньше или равен operandу справа.	var1 <= var2 var2 <= 5

WАрифметические операторы



Остаток от деления (%)	Бинарный оператор. Возвращает целочисленный остаток от деления двух operandов.	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.

WАрифметические 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.

Логические операторы

A C
W M

&&	expr1 && expr2	(Логическое И) Возвращает operand expr1 если он может быть преобразован в false; в противном случае возвращает operand expr2. Таким образом, при использовании булевых величин в качестве operandов, оператор && возвращает true если оба operandы true; в противном случае возвращает false.
	expr1 expr2	(Логическое ИЛИ) Возвращает operand expr1 если он может быть преобразован в true; в противном случае возвращает operand expr2. Таким образом, при использовании булевых величин в качестве operandов, оператор возвращает true если один из operandов true; если же оба false, то возвращает false.
!	!expr	(Логическое НЕ) Возвращает false если operand может быть преобразован в true; в противном случае возвращает true.

Условный оператор



```
var year = prompt('В каком году появилась спецификация ECMA-262 5.1?', '');

if (year < 2011) {
    alert( 'Это слишком рано...' );
} else if (year > 2011) {
    alert( 'Это поздновато...' );
} else {
    alert( 'Да, точно в этом году!' );
}
```

Тернарный оператор

A	C
W	
D	M

1 условие ? значение1 : значение2

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

A

W

Тернарный оператор

A	C
W	
D	M

```
var age = prompt('возраст?', 18);
```

```
var message = (age < 3) ? 'Здравствуй, малыш!' :  
(age < 18) ? 'Привет!' :  
(age < 100) ? 'Здравствуйте!' :  
'Какой необычный возраст!';
```

```
alert( message );
```

```
if (age < 3) {  
    message = 'Здравствуй, малыш!';  
} else if (age < 18) {  
    message = 'Привет!';  
} else if (age < 100) {  
    message = 'Здравствуйте!';  
} else {  
    message = 'Какой необычный возраст!';  
}
```



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.

A

W

While

A C
W M
D

```
while (condition) {  
    // код  
    // также называемый "телом цикла"  
}
```

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

A

W

while

A C
W M
D

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

A

W

for

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

A	C
W	
D	M

```
for (let i = 0; i < 3; i++) { // выведет 0, затем 1, затем 2  
    alert(i);  
}
```

A

W

break / continue

A	C
W	
D	M

```
for (let i = 0; i < 10; i++) {  
  
    // если true, пропустить оставшуюся часть тела цикла  
    if (i % 2 == 0) continue;  
  
    alert(i); // 1, затем 3, 5, 7, 9  
}
```

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».

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

A

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);



ФУНКЦИИ



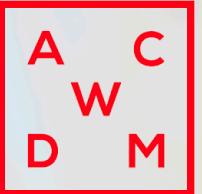
Зачастую нам надо повторять одно и то же действие во многих частях программы.

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.



ФУНКЦИЯ



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.

A



WФункциональное выражение

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

«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.

A

A C
W M
D

Функциональное выражение

```
// Function Declaration
function sum(a, b) {
    return a + b;
}
```

```
// Function Expression
var sum = function(a, b) {
    return a + b;
}
```

W

Параметры функции

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

A	C
W	
D	M

```
function showMessage(from, text) { // параметры from, text  
    from = "** " + from + " **"; // здесь может быть сложный код оформления  
    alert(from + ': ' + text);  
}  
  
showMessage('Маша', 'Привет!');  
showMessage('Маша', 'Как дела?');
```

A

Возврат значений

A	C
W	
D	M

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

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

A

W

Область видимости



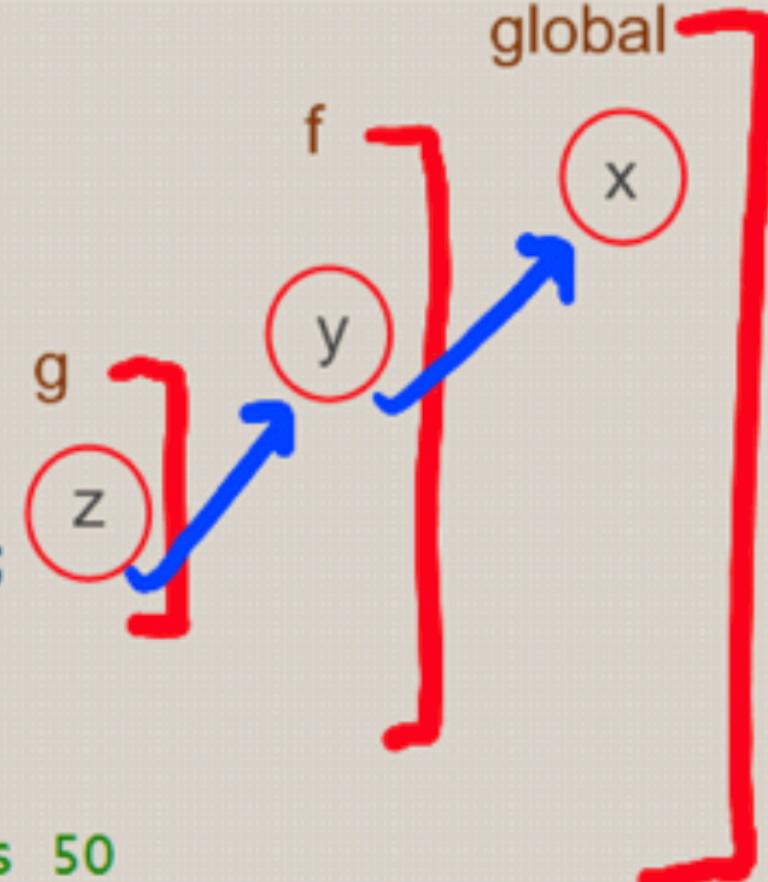
Функция - единственная конструкция в JS которая образует локальную область видимости. То есть переменные объявленые внутри функции не будут доступны наружу

W

Область видимости

A	C
W	
D	M

```
var x = 10;  
  
function f()  
{  
    var y = 15;  
  
    function g()  
    {  
        var z = 25;  
        alert(x+y+z);  
    }  
    g();  
}  
  
f(); // this displays 50
```

**A**



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.

```
function sayHi(name) {  
    // LexicalEnvironment = { name: 'Вася', phrase: undefined }  
    var phrase = "Привет, " + name;  
    alert( phrase );  
}  
  
sayHi('Вася');
```

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.»

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

Поднятие переменных

Что выведет следующий код ?

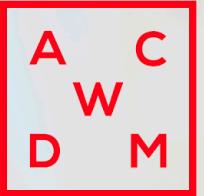
```
var foo = 1;  
  
function bar() {  
  
    if (!foo) {  
  
        var foo = 10;  
  
    }  
  
    alert(foo);  
  
}  
  
bar();
```



A



Замыкание



Замыкание – это функция вместе со всеми внешними переменными, которые ей доступны. Обычно, говоря «замыкание функции», подразумевают не саму эту функцию, а именно внешние переменные.



Замыкание

A C
W M
D

```
const counter = createCounter();
counter();counter();
```

```
const counter2 = createCounter(5);
counter2();counter2();
```

```
function createCounter(start){
    let count = start || 0;
    return function() {
        count++;
        console.log(count);
    }
}
```