JAVASCRIPT

LANGUAGE

EXTERNAL TRAINING .NET/WEB

EPAM SARATOV · SUMMER 2020

ПЛАН ЗАНЯТИЯ

- Ещё немного синтаксиса;
- Объекты;
- Массивы;
- Регулярные выражения;
- Конструкции;
- Функции;
- Стиль написания.

ИДЕНТИФИКАТОРЫ

- Могут содержать буквы (символы), цифры, **$** и **_**;
- Являются регистрозависимыми;
- Не могут начинаться с цифры;
- Могут содержать символы любого языка из Unicode.

```
//Корректные имена
var aBcD1 = 5;
var AbCd1 = 10;
var $ = 15;
var Не_делайте_так = 23;
```

```
//Некорректные имена
var 1aBcD = 5;
var Ab-Cd1 = 10;
```

ЗАПРЕЩЁННЫЕ ИДЕНТИФИКАТОРЫ

- Ключевые слова
  - break case catch continue default delete do else finally for function if in instanceof new return switch this throw try typeof var void while with
- Зарезервированные слова (на будущее)
  - abstract boolean byte char class const debugger double enum export extends final float goto implements import int interface long native package private protected public short static super synchronized throws transient volatile

```
var class = 5;
alert(class + 5);
```

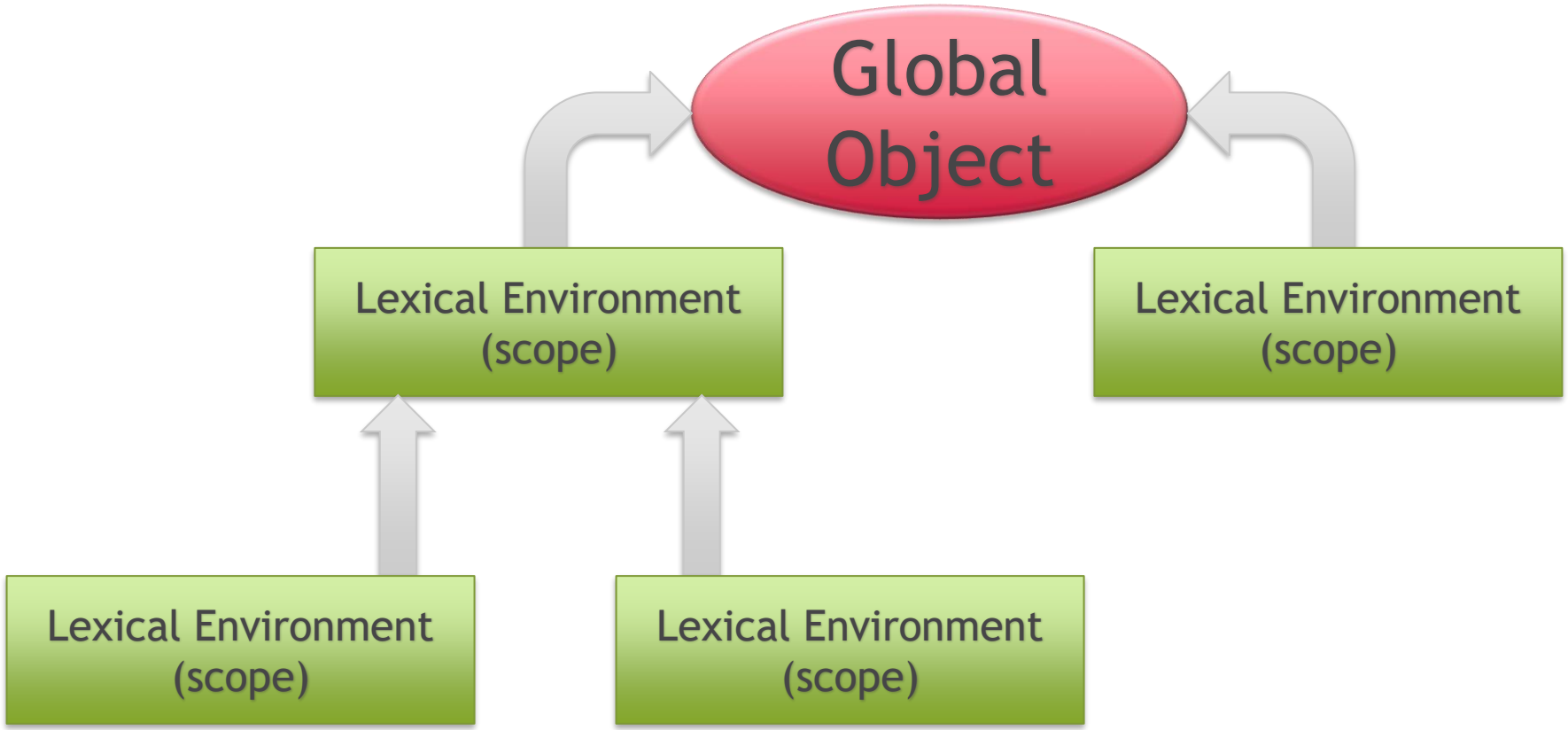
ТОЧКА С ЗАПЯТОЙ

- Если команда заканчивается переходом на новую строку, точку с запятой можно не ставить: она будет неявно добавлена туда автоматически;
- Но в некоторых важных ситуациях JavaScript делает это некорректно:
  - «забывает» вставить точку с запятой там, где она нужна;
  - ставит там, где она не нужна;
- Поэтому в JavaScript рекомендуется ставить точки с запятой всегда.

```
alert('Error!')
[1, 2, 3].toString()
```

```
function f() {
  return
        4;
}

alert(f());
```

ИЕРАРХИЯ JAVASCRIPT



ПРИВЕДЕНИЕ ТИПОВ

- Приведение к строке:
  - alert(**value**);
  - Сложение оператором **+** со строкой (даже с пустой)
- Приведение к числу:
  - Операции сравнения (кроме **===** и **!==**);
  - Унарный плюс **+**;
  - Арифметические операции;
- Приведение к логическому типу:
  - Использование в логическом контексте (**if**, **while**, **for**, **…**);
  - Двойное отрицание **!!**

ЧТО БУДЕТ В РЕЗУЛЬТАТЕ ПРЕОБРАЗОВАНИЯ?

- "" + 1 + 0 = 10
- "" - 1 + 0 = -1
- true + false = 1
- 6 / "3" = 2
- "2" * "3" = 6
- 4 + 5 + "px" = "9px"
- "$" + 4 + 5 = "$45"
- "4" – 2 = 2
- "4px" – 2 = NaN
- 7 / 0 = Infinity
- " -9\n" + 5 = " -9\n5"
- " -9\n" - 5 = -14
- 5 && 2 = 2
- 2 && 5 = 5
- 5 || 0 = 5
- 0 || 5 = 5
- null + 1 = 1
- undefined + 1 = NaN
- null == "\n0\n" = false
- +null == "\n0\n" = true

ССЫЛОЧНЫЕ И ЗНАЧИМЫЕ ТИПЫ

ЗНАЧИМЫЕ ТИПЫ ДАННЫХ

- Передаются по значению;
- Нельзя динамически расширять новыми свойствами/методами;
- Имеют объекты-обёртки;

- Типы:
  - Number
  - String
  - Boolean
  - Undefined
  - Null

ССЫЛОЧНЫЕ ТИПЫ ДАННЫХ

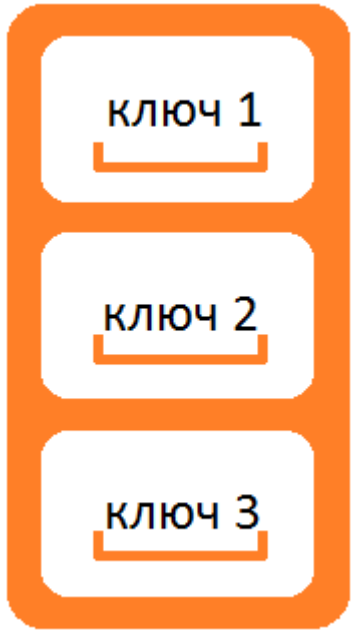
- Хранятся как набор пар ключ-значение;
- Передаются по ссылке;
- Можно динамически добавлять/удалять свойства и методы;

- Тип: **Object**. В частности:
  - Массивы
  - Дата
  - Регулярные выражения
  - Функции
  - Пользовательские объекты

ОБЪЕКТЫ

- Объекты в JavaScript реализованы как ассоциативные массивы: структуры, пригодные для хранения любых данных в виде пар ключ-значение.

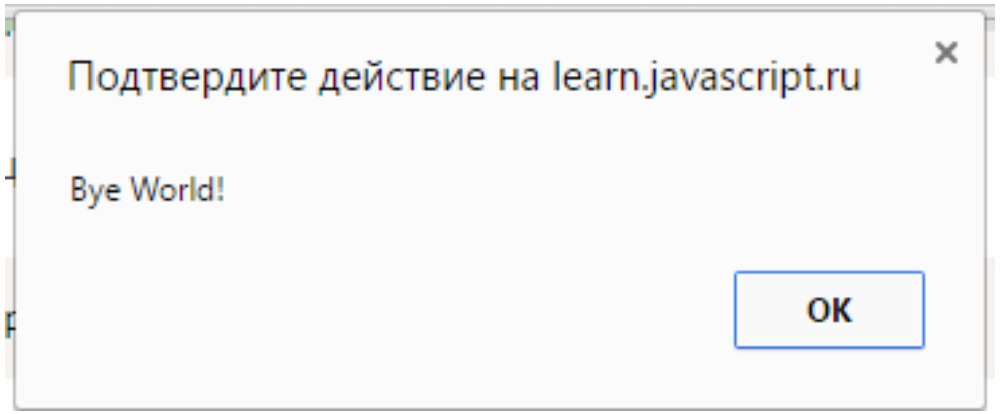
```
var object = new Object();
var object = {};

var object = {
        a : "a",
        "var" : "var",
        'field for empty' : "_",
        law : true
};
```



ПЕРЕДАЧА ПО ССЫЛКЕ

- В переменной, которой присвоен объект, хранится не сам объект, а «адрес в памяти», т.е. ссылка на него;
- При копировании переменной с объектом копируется только ссылка, но не сам объект.

```
var obj1 = {
  prop: "Hello World"
};
obj2 = obj1;
obj1.prop = "Bye World!";
alert(obj2.prop);
```



ИЗМЕНЕНИЕ ОБЪЕКТОВ

- Поскольку JavaScript является языком с динамической типизацией, он позволяет изменять не только тип данных, но и состав объектов: набор свойств и методов.

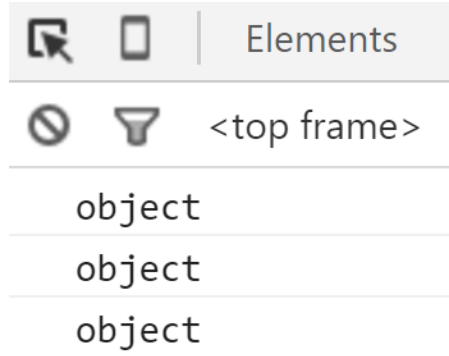
```
var obj = {};
obj.greeting = "Hello World";

console.log(obj.greeting);
obj.parting = "Bye World";
console.log(obj.parting);
```

ФУНКЦИИ-ОБЁРТКИ

- В JavaScript существуют функции-обёртки (конструкторы):
  - **Number()**, **Boolean()**, **String()**, **Null()**, **Undefined()**;
- Их задача — создать объект из примитива;
- Важно: результат будет объектом, а не значением примитивного типа.

```
var o1 = new Number(1),
    o2 = new String("1"),
    o3 = new Boolean(true);
```

```
console.log(typeof o1);
console.log(typeof o2);
console.log(typeof o3);
```



МАССИВЫ

- Массив — разновидность объекта, которая хранит пронумерованные значения и предлагает дополнительные методы для работы с ними;
- Особенности:
  - Является объектом;
  - Не типизирован;
  - Длина может изменяться;
  - Индексы (ключи) конвертируются в строки.

```
var array = new Array(1, 2, 3);
```

```
var array = [1, 2, 3];
```

```
var array = [1, "232423", true, [1, 2]];
```

КОНСТРУКТОР МАССИВОВ

- Конструктор массива не рекомендуется к использованию из-за его неоднозначного поведения:
  - При передаче более одного параметра происходит инициализация массива переданными значениями.
  - При передаче одного параметра происходит инициализация размера массива переданным параметром.

```
var a = new Array(1, 2);
console.log(a);
var a1 = new Array(2);
console.log(a1);
var a2 = new Array(2.7);
console.log(a2);
```

```
[1, 2]
[]
▶ Uncaught RangeError: Invalid array length
```

МАССИВЫ. ДОСТУП К ЭЛЕМЕНТАМ.

- Массив является объектом, и каждое значение в нем хранится как пара ключ — значение;
- Доступ к элементам реализуется так же, как и в ассоциативном массиве;
- Внутри массива могут содержаться элементы любых типов (в том числе и другие массивы).

```
var array = ["строка", 10, true, [1, 10]];
console.log(array[2]);
console.log(array["0"]);
console.log(array[3][1]);
```

```
true
строка
10
```

МАССИВЫ. РАЗРЕЖЕННЫЙ МАССИВ.

- Массивы бывают разреженными;
- Массив ведёт себя как массив при идущих подряд индексах;
- Если записать в массив элемент по несуществующему индексу, массив автоматически расширится, задав остальным элементам значение **undefined**.

```
var array = [0, 1, 2];
array[3] = 3;
console.log(array);
array[10] = 10;
console.log(array);
console.log(array[5]);
```

```
[0, 1, 2, 3]
[0, 1, 2, 3, 10: 10]
undefined
```

МАССИВЫ. LENGTH

- Изменить длину массива также можно, явно задав свойству **length** нужное значение;
- Незаданные элементы получат значение **undefined**.

```
var array = [0, 1, 2];
console.log(array);
console.log(array.length);
array.length = 5;
console.log(array);
console.log(array.length);
console.log(array[4]);
```

```
[0, 1, 2]
3
[0, 1, 2]
5
undefined
```

ДОБАВЛЕНИЕ И УДАЛЕНИЕ ЭЛЕМЕНТОВ ИЗ МАССИВА

Метод	Описание
unshift(элемент)	Добавляет элемент в начало массива
shift()	Удаляет элемент из начала массива и возвращает его к качестве результата работы
push(элемент)	Добавляет элемент в конец массива
pop()	Удаляет элемент из конца массива и возвращает его к качестве результата работы

- Используя данные методы, мы можем работать с массивом как со стеком или очередью.

МЕТОДЫ МАССИВА: SPLIT

- Метод split разбивает строку по разделителю и возвращает массив элементов.
  - **.split(**[разделитель], [количество элементов в массиве]**)**
- Особенности:
  - Если передать в метод второй параметр, то вернётся не более указанного числа элементов (лишние будут отброшены);
  - Если ничего не передать, вернётся массив с одним элементом.

```
console.log("1,2,3,4,5".split(','));          ["1", "2", "3", "4", "5"]
console.log("1,2,3,4,5".split(',', 2));       ["1", "2"]
console.log("1,2,3,4,5".split());             ["1,2,3,4,5"]
```

МЕТОДЫ МАССИВА: JOIN

- Метод **join(**разделитель**)** выполняет задачу, обратную **split:** он склеивает элементы массива в одну строку, вставляя между элементами указанный разделитель;
- Если разделитель не указан, по умолчанию используется запятая.

```
var array = [0, 1, 2];
console.log(array.join("|"));
console.log(array.join());
```

```
0|1|2
0,1,2
```

МЕТОДЫ МАССИВА: SORT

- Метод **sort()** производит упорядочивание массива;
- Особенности:
  - По умолчанию сортировка элементов происходит через сравнение строковых представлений элементов;
  - Для указания своего порядка сортировки нужно передать функцию сравнения.

```
var array = [10, 6, 0, 1, 4, 2];
console.log(array);
console.log(array.sort());
function compare(a,b) {
    return b - a;
}
console.log(array.sort(compare));
```

```
[10, 6, 0, 1, 4, 2]
[0, 1, 10, 2, 4, 6]
[10, 6, 4, 2, 1, 0]
```

МЕТОДЫ МАССИВА: CONCAT

- Создает новый массив, состоящий из элементов текущего и дополненный переданными аргументами;
- Если в качестве аргумента передать массив, то добавятся его элементы.

```
var arr1 = [1, 2, 3, 4],
    arr2 = [6, 7, 8, 9, 0],
    arr;

arr = arr1.concat(5, arr2);

console.log(arr);
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
```

МЕТОДЫ МАССИВА: DELETE

- Метод **delete** производит удаление элемента из массива;
- После удаления элемента его ключу ставится в соответствие undefined;
- Не рекомендуется к использованию.

```
var arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 0];
console.log(arr);
delete arr[3];
console.log(arr);
console.log(arr[3]);
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 0]

[1, 2, 3, 4: 5, 5: 6, 6: 7, 7: 8, 8: 9, 9: 0]
undefined
```

МЕТОДЫ МАССИВА

Методы/свойства	Описание
reverse()	Инвертирует порядок элементов в массиве.
slice([начало[,конец]])	Возвращает подмассив, ограниченный указанными индексами.
splice(начало,число[,v1…vn])	Удаляет из массива часть элементов и заменяет их аргументами v
indexOf(элемент[,индекс])	Находит индекс первого вхождения элемента в массиве, начиная с указанного индекса (если он передан).
lastIndexOf(элемент[,индекс])	Находит индекс последнего вхождения элемента с конца массива начиная с указанного индекса (если он передан).

ДАТА

- Для работы с датой и временем в JavaScript используются объект Date.
- Существует 4 способа обращения к конструктору объекта Date:
  - new Date() — создаёт объект Date с текущими датой и временем;
  - new Date(число) — создаёт объект Date, значение которого равно количеству миллисекунд, прошедших с 1 января 1970 года GMT+0;
  - new Date(строка) — создаёт объект Date, разбирая строковое представление даты;
  - new Date(год, месяц, число [, часы, минуты, секунды, миллисекунды]) — создаёт объект Date, используя компоненты в локальной временной зоне.

ДАТА: СТРОКОВОЕ ПРЕДСТАВЛЕНИЕ

YYYY-MM-DDTHH:mm:ss.sssZ — `2000-12-12T12:12:12.000+04:00`

Компонент	Описание
YYYY	год по григорианскому календарю.
MM	месяц года с 01 (январь) по 12 (декабрь).
DD	день месяца с 01 по 31.
T	буквальное изображение “T” обозначает начало элемента время.
HH	количество полных часов, прошедших с полуночи
mm	количество полных минут
ss	количество полных секунд
sss	количество полных миллисекунд
Z	смещение на часовой пояс, указанное в виде “Z” (для UTC)

ДАТА: ФОРМАТЫ ВЫВОДА ДАТЫ.

```
var date = new Date("2000-12-12T12:12:12.000+04:00");
```

Метод	Результат
date.toString();	Tue Dec 12 12:12:12 UTC+0400 2000
date.toUTCString();	Tue, 12 Dec 2000 08:12:12 UTC
date.toGMTString();	Tue, 12 Dec 2000 08:12:12 UTC
date.toISOString();	2000-12-12T08:12:12.000Z
date.toLocaleString();	Tuesday, December 12, 2000 12:12:12 PM
date.toDateString();	Tue Dec 12 2000
date.toLocaleDateString();	Tuesday, December 12, 2000
date.toTimeString();	12:12:12 UTC+0400
date.toLocaleTimeString();	12:12:12 PM

ДАТА: МЕТОДЫ

Получение	Установка	Компонент даты
getTime()	setTime(n)	Число миллисекунд с 1 января 1970 г.
getFullYear()	setFullYear(n)	Год полностью
getMonth()	setMonth(n)	Месяц (начиная с 0)
getDate()	setDate(n)	Число
getHours()	setHours(n)	Часы
getMinutes()	setMinutes(n)	Минуты
getSeconds()	setSeconds(n)	Секунды
getMilliseconds()	setMilliseconds(n)	Миллисекунды
getDay()		День недели (Воскресенье = 0)

РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ

- Регулярные выражения в JavaScript имеют некоторые особенности;
- Для создания регулярного выражения используются следующие форматы:
  - Конструктор regex;
  - Строчное описание;
- При обоих видах описания в JavaScript можно конфигурировать параметры работы указывая флаги.

```
var reg = /\d+?/;
var reg = /\d+/i;
var reg = new RegExp("\\d+?");
var reg = new RegExp("\\d+", "i");
```

Флаг	Свойство	Описание
i	ignoreCase	Поиск не чувствительный к регистру
g	global	Находит все соответствия
m	multiline	Многострочный режим

РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ: СВОЙСТВА И МЕТОДЫ

Методы/Свойства	Описание
source	Возвращает строку с регулярным выражением
lastIndex	Указывает на позицию в строке, с которой начинается следующий поиск совпадения
exec(string)	Производит поиск соответствия регулярному выражению в строке и возвращает массив, содержащий результаты соответствия, или null, если соответствие не найдено.
test(string)	Возвращает true, если найдено соответствие

МЕТОДЫ СТРОК С РЕГУЛЯРНЫМИ ВЫРАЖЕНИЯМИ

Метод	Описание
match(regexp)	Возвращает массив, содержащий все совпадения.
search(regexp)	Возвращает индекс первого вхождения совпадения или -1, если совпадения не найдены.
replace(regexp, замена)	Возвращает строку, в которой все совпадения заменены на строку замена.
split(regexp)	Возвращает массив подстрок, разделяемых подстроками, удовлетворяющими регулярному выражению.

УСЛОВНЫЕ ОПЕРАТОРЫ

- В JavaScript существуют следующие условные операторы:
  - оператор if
  - оператор ?
- Оператор if(“если”) получает условие, вычисляет его и, если условие выполняется, происходит выполнение команды.

```
if (age < 14) {
    alert('Вы не можете зарегистрироваться.');
}
```

- Оператор if (...) при вычислении значения в скобках производит преобразование к логическому типу.
- При преобразовании:
  - 0, '', NaN, null, undefined – false
  - Остальные значения – true

УСЛОВНЫЕ ОПЕРАТОРЫ: IF

- При использовании оператора if мы также можем использовать необязательный блок else.

```
if (age < 14) {
    alert('Вы не можете зарегистрироваться.');
}
else {
    alert('Добро пожаловать!');
}
```

- При необходимости можно объединять несколько конструкций if else в цепочки проверяемых условий.

```
if (age < 14) {
    alert('Вы не можете зарегистрироваться.');
}
else if(age==14){
    alert('Вам точно есть 14?');
}
else {
    alert('Добро пожаловать!');
}
```

УСЛОВНЫЕ ОПЕРАТОРЫ: ?:

- Иногда может понадобиться в зависимости от контекста задачи сократить вариант записи конструкции if else. Для этого можно использовать оператор ?, который позволяет делать это короче.

```
if (age > 14) {
    access = true;
} else {
    access = false;
}

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

SWITCH

- Конструкция switch позволяет заменить способ замены цепочки проверок if else.
- Особенности:
  - Если break нет, то выполнение пойдёт ниже по следующим case, при этом остальные проверки игнорируются.
  - В case могут быть любые выражения:
    - переменные
    - функции
  - При сравнение значения в case и expression происходит приведение типа

```
var a = 1;
switch (a) {
    case 1: {
        alert("Один");
    } break;
    case 2: {
        alert("Два");
    } break;
    case 3: {
        alert("Три");
    } break;
    default: {
        alert("Другое значение");
    } break;
}
```

ЦИКЛЫ: FOR

- Цикл for предназначен для выполнения описанного блока кода заданное пользователем количество раз. Цикл for имеет следующий синтаксис:

```
for ([инициализация];[условие];[действие]){
   тело цикла
}
```

Особенности:

- Все блоки являются необязательными
- Блок инициализации выполняется один раз
- Условие проверяется перед каждой итерацией
- Действие выполняется после каждой итерации

```
for (var i = 0; i < 10; i++) {
    alert(i);
}
```

ЦИКЛЫ: WHILE

- Цикл while – предназначен для выполнения описанного блока кода, пока выполняется условие. Цикл while имеет следующий синтаксис:

```
while(условие) {
    тело цикла
}
```

```
var i = 0;
while (i < 10) {
    alert(i);
    i++;
}
```

ЦИКЛЫ: DO...WHILE

- Цикл do..while – предназначен для выполнения описанного блока кода, пока выполняется условие. Цикл do..while имеет следующий синтаксис:

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

```
var i = 0;
do{
    alert(i);
    i++;
} while (i < 10)
```

- В отличии от цикла while, данный цикл выполнит как минимум одну итерацию.

BREAK И CONTINUE

- break – полностью прекращает выполнение цикла и передаёт управление на строку за его телом.

```
var i = 0;
while (i < 10) {
    alert(i);
    i++;
    if (i === 5) {
        break;
    }
}
```

- continue — прекращает выполнение текущей итерации цикла и передает управление следующей итерации.

```
var i = 0;
while (i < 10) {
    alert(i);
    i++;
    if (i === 5) {
        continue;
    }
}
```

ЦИКЛЫ: FOR IN

- Для перебора всех свойств из объекта используется цикл по свойствам for..in. Эта синтаксическая конструкция отличается от рассмотренного ранее цикла for(;;).
- Особенности:
  - Порядок перебора свойств строго не задан и может различаться в различных браузерах
  - Key – это имя свойства а не значения, обращение к значению происходит используя конструкцию obj[key];

```
for (key in obj) {
    /* ... делать что-то с obj[key] ... */
}
```

ЦИКЛЫ: ПРАВИЛЬНОЕ ИСПОЛЬЗОВАНИЕ FOR IN

```
var person = {
    name: "Ivan",
    surname: "Ivanov",
    age: 16
};
```

- Для корректной работы цикла for in необходимо использовать метод .hasOwnProperty(prop) перебираемого объекта, чтобы исключить свойства, полученные по линии наследования.

```
for (var key in person) {
    if (person.hasOwnProperty(key)) {
        alert(person[key]);
    }
}
```

```
var array = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

for (var key in array) {
    if (array.hasOwnProperty(key)) {
        alert(array[key]);
    }
}
```

TRY CATCH

- Для перехвата и обработки ошибок в JavaScript присутствует конструкция try catch.
- Особенности:
  - В блоке try пишется код в котором возможны ошибки.
  - Если ошибок нет, блок catch пропускается
  - Если есть ошибки, выполнение блока try прерывается и начинается выполнение блока catch

```
function func() {
    throw 12345;
}

try {
    func();
} catch (e) {
    if (e == 12345) {
        alert("Error 12345")
    }
    else {
        alert("Unknow error");
    }
}
```

TRY CATCH FINALLY

- Конструкция try catch может содержать блок finally.

- Секция finally не обязательна, но, если она есть, то она выполняется всегда:
  - После блока try, если ошибок не было
  - После блока catch если ошибка была

```
function func() {
    throw 12345;
}
try {
    func();
} catch (e) {
   alert("Error")
}
finally {
    alert("Always run");
}
```

ФУНКЦИИ

- Функции позволяют нам выполнять одно и тоже действие в разных местах программы.
- Особенности:
  - Функция создает область видимости
  - Количество входных аргументов при вызове может не совпадать с количеством аргументов при описании.
  - Если аргумент не передан, он считается undefined
  - Функция может вернуть результат своей работы с помощью оператора return.
  - Если в функции отсутствует return, возвращается undefined.
  - Отсутствует перегрузка функций

```
function func() {
    alert("Hello World!");
}
```

ПРАВИЛЬНОЕ НАПИСАНИЕ ФУНКЦИЙ

- Имя функции следует тем же правилам, что и имя переменной.
- По правилам написания названия функции имя должно быть глаголом, т.к. функция — это действие.
- Функция должна делать то, что подразумевает её название
- Одна функция — одно действие (SRP)

```
showResult();
getElement();
createBlock();
```

ШАБЛОН МОДУЛЬ

- Чтобы у скрипта была своя область видимости и не происходило добавление переменных в глобальную область видимости, применяется шаблон “Модуль”.

- Т.к. функция создаёт свою область видимости, все наши переменные будут находиться в свой области видимости и не будут конфликтовать с другими скриптами.

```
(function () {
    var text;

    function Hi() {
        text="Hello world!"
    }

    Hi();
})();
```

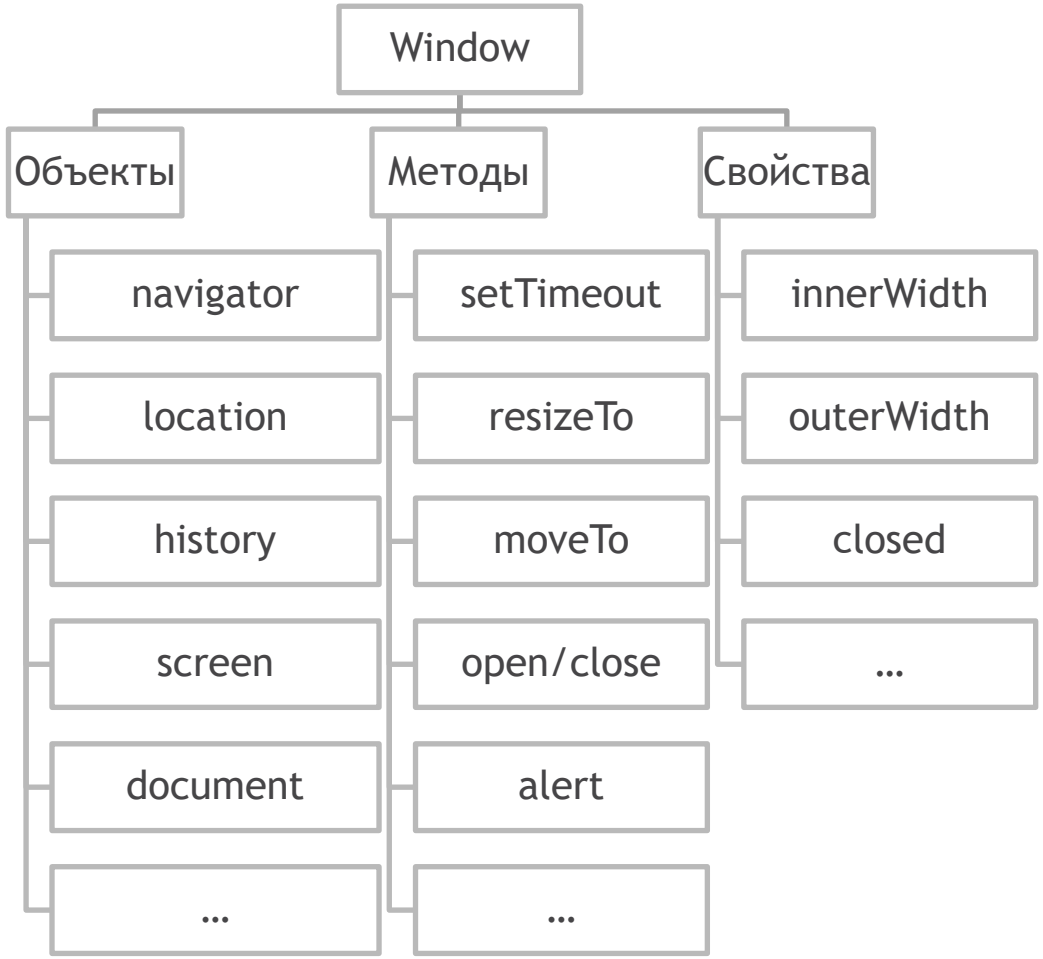
ПРАВИЛЬНОЕ НАПИСАНИЕ ОБЪЯВЛЕНИЯ ПЕРЕМЕННЫХ

- Так как переменные в JavaScript поднимаются в начало, имеет смысл описывать объявление переменных в начале функции или файла.

- Для более логичного структурирования JavaScript кода принято соглашение описать все объявления переменных с помощью одного var.

```
var a = 1,
    b,
    c,
    result;
```

BOM. WINDOW. ГЛОБАЛЬНЫЙ ОБЪЕКТ.



BOM. ТАЙМЕРЫ

Функции	Описание
setTimeout(функция[,задержка[,аргументы]])	Асинхронно запускает функцию
clearTimeout(ссылка)	Останавливает запуск функции, запущенной через setTimeout
setInterval(функция[,период[,аргументы]])	Периодически вызывает функцию.
clearInterval(ссылка)	Останавливает запуск функции, запущенной через setInterval

BOM. ИСТОРИЯ (HISTORY)

Метод/Свойство	Описание
length	Количество записей в текущей сессии истории
state	Возвращает текущий объект истории (HTML5)
go(n)	Метод, позволяющий перемещаться по истории. n - смещение, относительно текущей позиции.
back()	Метод, идентичный вызову go(-1)
forward()	Метод, идентичный вызову go(1)
pushState(data, title [, url])	Добавляет элемент истории (HTML5)
replaceState(data, title [, url])	Обновляет текущий элемент истории (HTML5)
popstate	Событие возникающее при переходе по истории (HTML5)

BOM. РАСПОЛОЖЕНИЕ (LOCATION)

Свойство/Метод	Описание
hash	часть URL, которая идет после символа решетки ‘#’
host	хост и порт
href	весь URL. Можно менять
hostname	хост (без порта)
pathname	строка пути (относительно хоста)
port	номер порта (если порт не указан, то пустая строка)
protocol	протокол
search	часть адреса после символа "?"
assign(url)	загружает документ по данному url
reload([кэш])	перезагружает документ по текущему URL. Аргумент кэш показывает, можно ли брать страницу из кэша
replace(url)	заменяет текущий документ на документ по указанному url

BOM. НАВИГАТОР (NAVIGATOR)

Свойство	Описание
appName	Имя браузера
appVersion	Версия браузера
platform	Операционная система
userAgent	Информация о клиенте
language	Язык
onLine	Показывает является ли запрос страницы online

BOM. ДИАЛОГИ

Метод	Описание
alert(сообщение)	Выводит модальное окно с сообщением.
confirm(сообщение)	Выводит модальное окно с двумя кнопками: "ОК" и "ОТМЕНА". Возвращает true/false
prompt(сообщение[, значение])	Выводит сообщение в окне с текстовым полем и двумя кнопками: "ОК" и "ОТМЕНА". Возвращает введенную строку или null
showModalDialog(uri[, аргументы])	Выводит страницу как модальное окно и возвращает значение

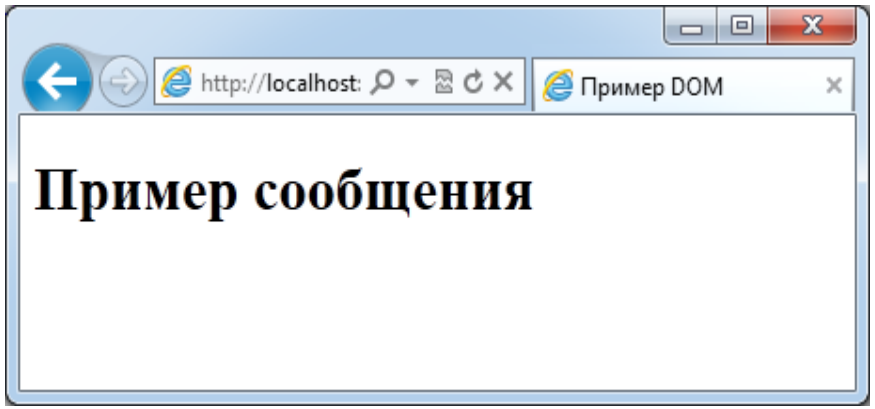
DOM

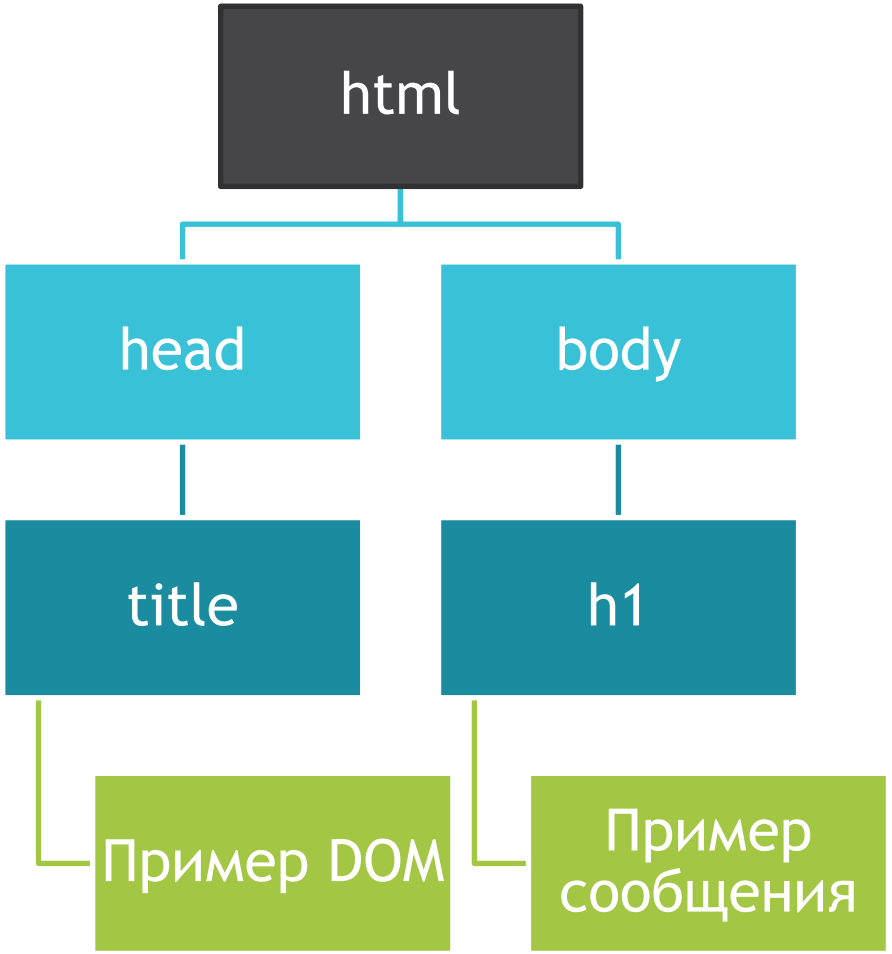
DOM (Document Object Model — «объектная модель документа») — это независящий от платформы и языка программный интерфейс, позволяющий программам и скриптам получить доступ к содержимому HTML, XHTML и XML-документов, а также изменять содержимое, структуру и оформление таких документов.

http://www.w3.org/DOM/

ПРИМЕР DOM

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
 <title>Пример DOM</title>
</head>
<body>
 <h1>Пример сообщения</h1>
</body>
</html>
```





DOM. NODE (УЗЕЛ)

Свойство	Описание
nodeType	Код типа узла
nodeName	Имя типа узла
nodeValue	Значение узла
ownerDocument	Ссылка на документ
attributes	Массив атрибутов узла

DOM. ELEMENT (ЭЛЕМЕНТ)

Метод/Свойство	Описание
tagName	Имя типа элемента
innerHTML	HTML-содержимое узла
outerHTML	HTML-содержимое узла вместе с самим элементом
innerText/textContent	Содержимое элемента в виде текста

DOM. ПОИСК ЭЛЕМЕНТОВ.

Метод	Описание
getElementById(id)	Поиск по id одного элемента
getElementByName(имя)	Поиск по атрибуту name элемента
getElementsByTagName(тэг)	Поиск по имени tag массива элементов
getElementsByClassName(класс)	Поиск по имени CSS класса массива элементов
querySelectorAll(селектор)	Поиск по CSS селектору массива элементов
querySelector(селектор)	Поиск по CSS селектору одного элемента

DOM. СОЗДАНИЕ ЭЛЕМЕНТОВ.

Метод	Описание
createElement(tagName)	Создает элемент заданного типа
createTextElement(text)	Создает текстовый элемент с заданным с заданным содержимым

DOM. ДОБАВЛЕНИЕ/УДАЛЕНИЕ УЗЛОВ

Метод	Описание
appendChild(node)	Добавляет node в список дочерних элементов. Новый узел добавляется в конец списка.
insertBefore(node, nextSibling)	Вставляет node в список дочерних элементов, перед узлом nextSibling
replaceChild(node, currentNode)	Среди детей заменяет currentNode на node.
removeChild(node)	Удаляет node из списка детей

DOM. АТРИБУТЫ.

Методы/Свойства	Описание
attributes	Набор атрибутов (тип Attr)
hasAttribute(name)	Проверяет наличие атрибута
getAttribute(name)	Получает значение атрибута
setAttribute(name, value)	Устанавливает значение атрибута
removeAttribute(name)	Удаляет атрибут

DOM. СТИЛИ.

Метод/Свойство	Описание
className	Содержимое атрибута class
classList	Список стилей (DOMTokenList)
style	Объект, содержащий значения из атрибута style (CSSStyleDeclaration)
getComputedStyle()	Возвращает все вычисленные стили конкретного элемента
getBoundingClientRect()	Возвращает координаты и размеры элемента в клиентской области браузера

DOM. СТИЛИ. DOMTOKENLIST

Методы/Свойства	Описание
length	Количество классов
item(индекс)	Получает имя класса по индексу
contains(класс)	Проверяет наличие класса
add(класс)	Добавляет класс
remove(класс)	Удаляет класс
toggle(класс)	Удаляет класс, если он есть, или добавляет его, если не было. Возвращает значение, показывающее наличие указанного класса после вызова метода

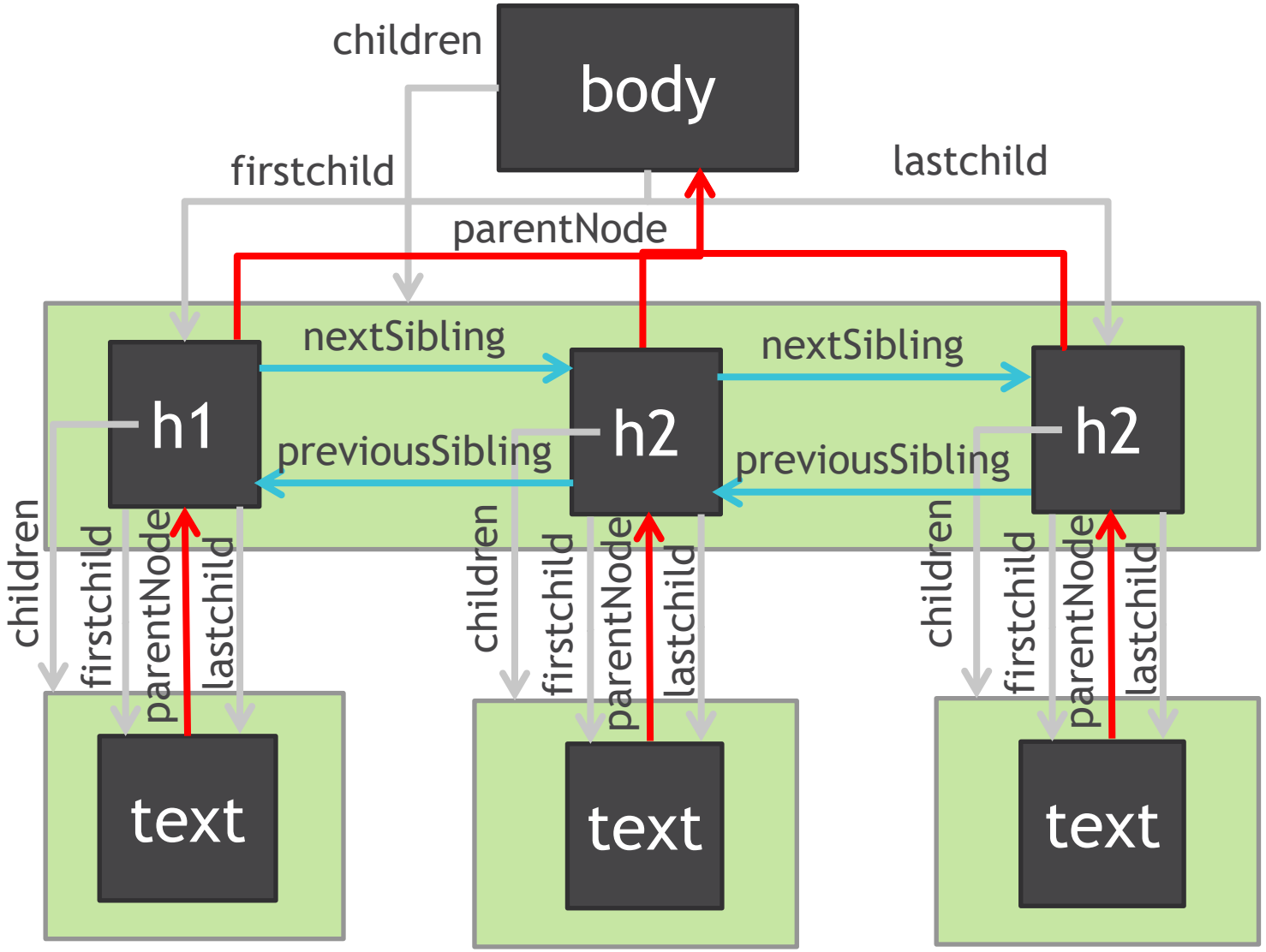
DOM. СТИЛИ. CSSSTYLEDECLARATION

Методы/Свойства	Описание
length	Количество свойств
cssText	Полный текст стиля
parentRule	Ссылка на правило, которому принадлежит стиль
item(индекс)	Получает имя свойства по индексу
getPropertyPriority(свойство)	Проверяет приоритет свойства
getPropertyValue(свойство)	Получает значение свойства
removeProperty(свойство)	Удаляет свойство
setProperty(свойство, значение, приоритет)	Устанавливает значение и приоритет свойства

DOM. ОБХОД

Свойство	Описание
documentElement	Тэг <html>
body	Тэг <body>
childNodes	Все дочерние узлы узла
children	Все дочерние элементы элемента
firstChild/lastChild	Первый/последний дочерний узел
parentNode	Родительский узел
nextSibling/previousSibling	Левый/правый сосед

DOM. ОБХОД


children
body
firstchild
lastchild
parentNode
nextSibling
nextSibling
h1
h2
h2
previousSibling
previousSibling
children
firstchild
parentNode
lastchild
children
firstchild
parentNode
lastchild
children
firstchild
parentNode
lastchild
text
text
text

СОБЫТИЯ. СТАНДАРТНЫЕ СОБЫТИЯ

События загрузки документа

- abort, error, readystatechange, load, unload

События клавиатуры

- keydown, keypress, keyup

События «мыши»

- mousedown, mouseenter, mouseleave, mousemove, mouseout, mouseover, mouseup, click, dblclick

События формы

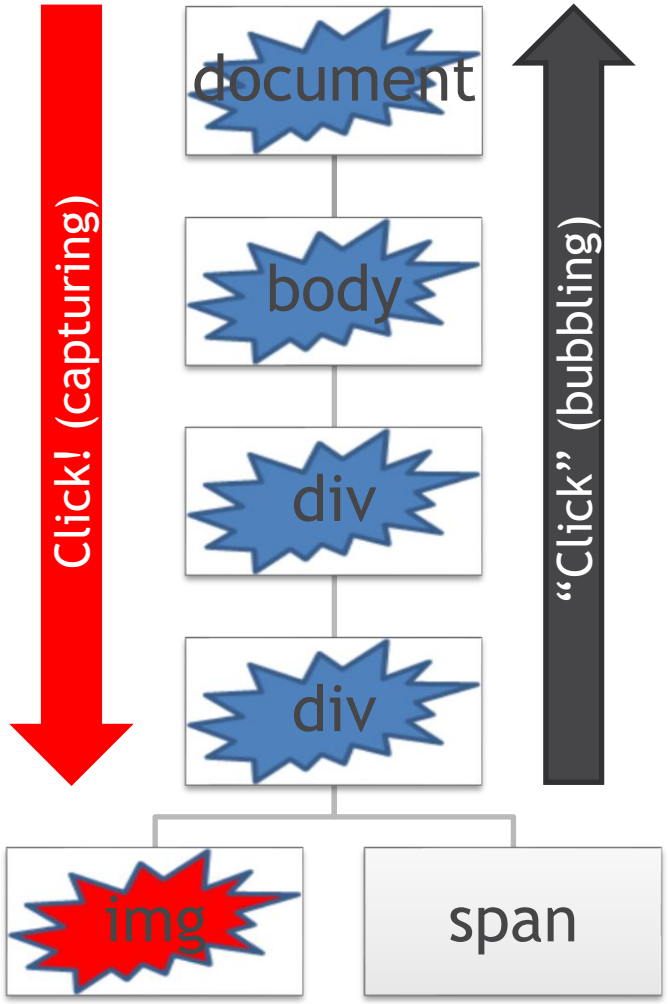
- select, focus, blur, focusin, focusout, reset, submit.

События окна

- resize, scroll

СОБЫТИЯ. ВСПЛЫТИЕ (BUBBLING)

```
<html>
<head>
 <title></title>
</head>
<body>
 <div onclick="alert('Корень');">
   <h2>Без обработчика</h2>
   <div onclick="alert('Кнопка');">
     <img src="filesave.png" />
     <span>Сохранить</span>
   </div>
 </div>
</body>
</html>
```



СОБЫТИЯ. ПОДПИСЫВАНИЕ.

В разметке

```
<div onclick="alert('Кнопка');">
 Сохранить
</div>
```

Через свойство:

```
window.onresize = GetSize;
```

Через «слушателя»:

```
window.addEventListener("resize", GetSize, true);
```

СОБЫТИЯ. ОБЪЕКТ СОБЫТИЯ

Метод/Свойство	Описание
type	Тип события
target	Объект, над которым произошло действие
currentTarget	Объект, перехвативший событие (this)
bubbles	Всплывающее событие
cancelable	Может быть отменено
eventPhase	Статус события
preventDefault	Отменяет действие по умолчанию
stopPropagation	Отменяет всплытие события.

СОБЫТИЯ. СОЗДАНИЕ СОБСТВЕННОГО СОБЫТИЯ

Новый синтаксис

```
var evt = new CustomEvent("buttonclick", { bubbles: true });
this.dispatchEvent(evt);
```

Старый синтаксис

```
evt = document.createEvent("Events");
evt.initEvent("buttonclick", true, true);
this.dispatchEvent(evt);
```

ФОРМЫ

Варианты обращения к формам

- window.sendForm;
- document.sendForm;
- document.forms.sendForm;
- document.forms[0];

Варианты обращения к элементам формы

- document.forms.sendForm[0];
- document.forms.sendForm.firstName;
- document.forms.sendForm.elements.firstName;
- document.forms.sendForm.elements[0];

```
<form id="sendForm">
 <input name="firstName" type="text" />
</form>
```

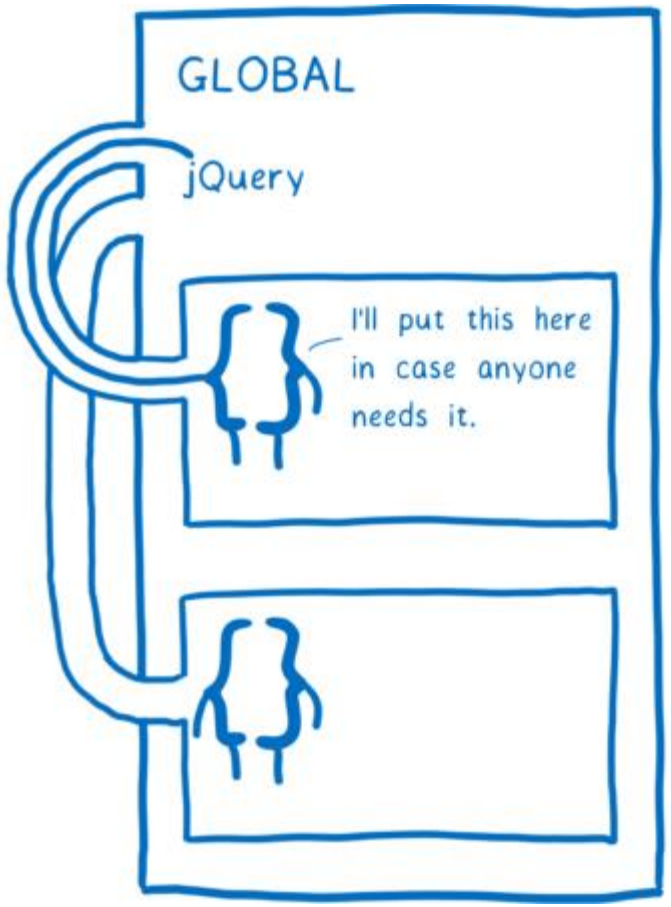
ФОРМЫ

Методы/свойства	Описание
acceptCharset	Устанавливает кодировку, в которой сервер может принимать и обрабатывать данные
action	Адрес программы, которая обрабатывает данные формы
autocomplete	Включает автозаполнение полей формы (HTML5)
enctype	Способ кодирования данных формы(application/x-www-form-urlencoded, multipart/form-data, text/plain)
method	Метод протокола HTTP
name	Имя формы
noValidate	Отменяет встроенную проверку данных формы (HTML5)
target	Имя окна или фрейма, куда загружается возвращаемый результат
elements	Содержит коллекцию элементов
submit()	Отправляет данные формы на сервер
reset()	Восстанавливает значения по умолчанию
checkValidity()	Проверяет валидность полей формы форм

OOP

```
class User {

  constructor(name) {
    this.name = name;
  }

  sayHi() {
    alert(this.name);
  }

}

// Использование:
let user = new User("Иван");
user.sayHi();
```

MODULES




epam
THANKS FOR ATTENTION!
VIKENTII EKGART & DANILA KALITA, SARATOV, RUSSIA