Современные веб-технологии

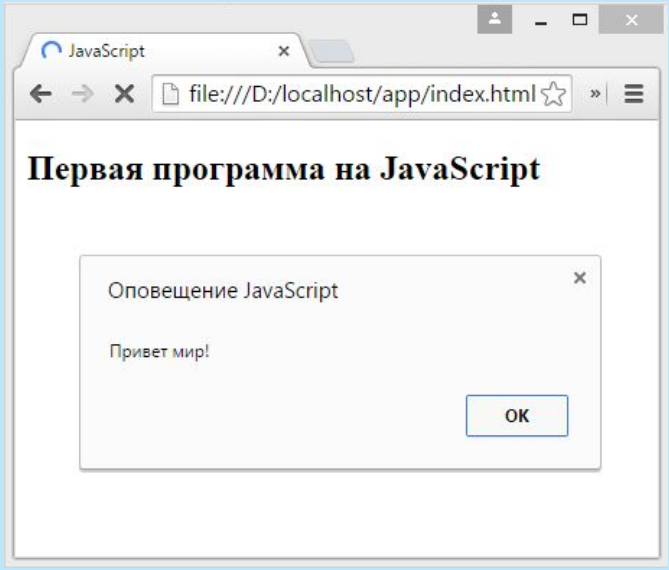
JavaScript

JavaScript

- JavaScript был создан в 1995 году в компании Netscape в качестве языка сценариев в браузере Netscape Navigator 2.
- Первоначально язык назывался LiveScript, но на волне популярности в тот момент другого языка Java LiveScript был переименован в JavaScript.

JavaScript

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>JavaScript</title>
</head>
<body>
    <h2>Первая программа на JavaScript</h2>
    <script>
    alert('Привет мир!');
    </script>
</body>
</html>
```



JavaScript

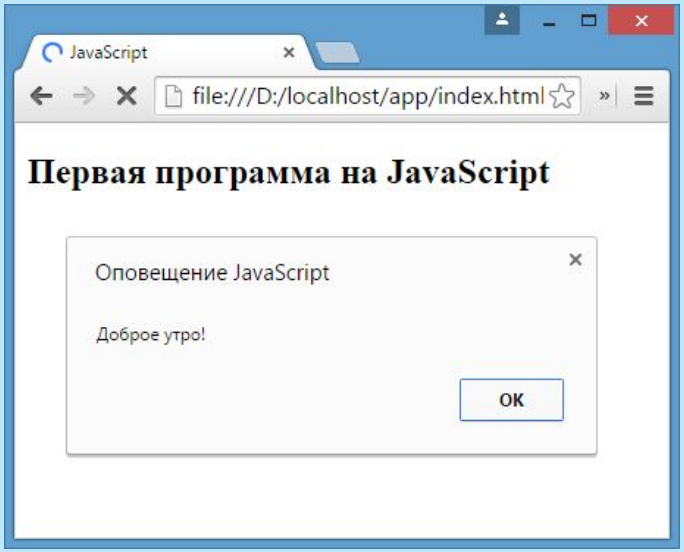
Еще один способ подключения кода JavaScript на веб-страницу представляет вынесение кода во внешние файлы и их подключение с помощью тега <script>

файл *myscript.js*

```
var date = new Date(); // получаем текущую дату
var time = date.getHours(); // получаем текущее время в часах
if(time < 13) // сравниваем время с число 13
    alert('Доброе утро!'); // если время меньше 13
else
    alert('Добрый вечер!'); // если время равно 13 и больше
```

JavaScript

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
<title>JavaScript</title>
</head>
<body>
    <h2>Первая программа на JavaScript</h2>
    <script src="js/myscript.js"></script>
</body>
</html>
```

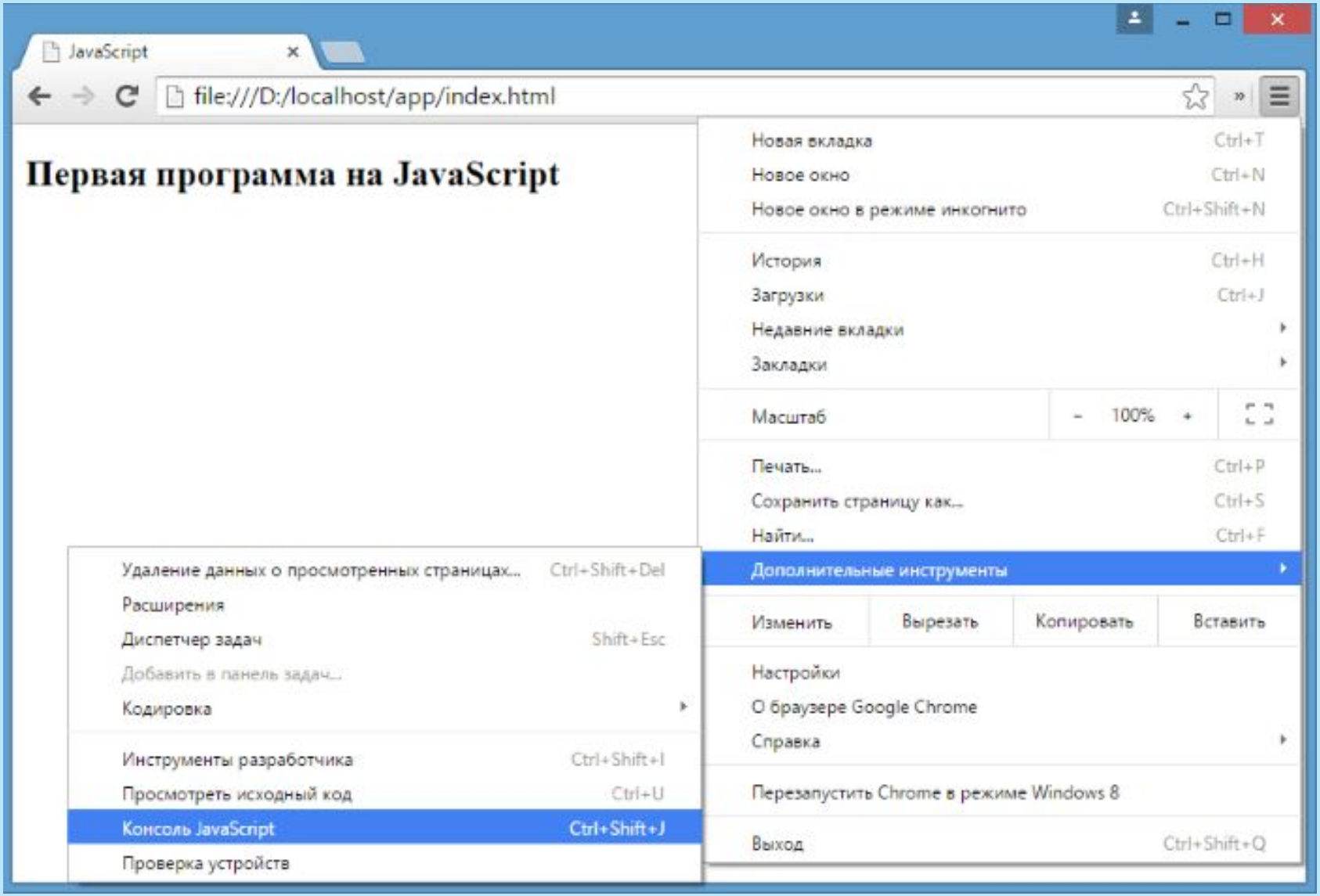


JavaScript

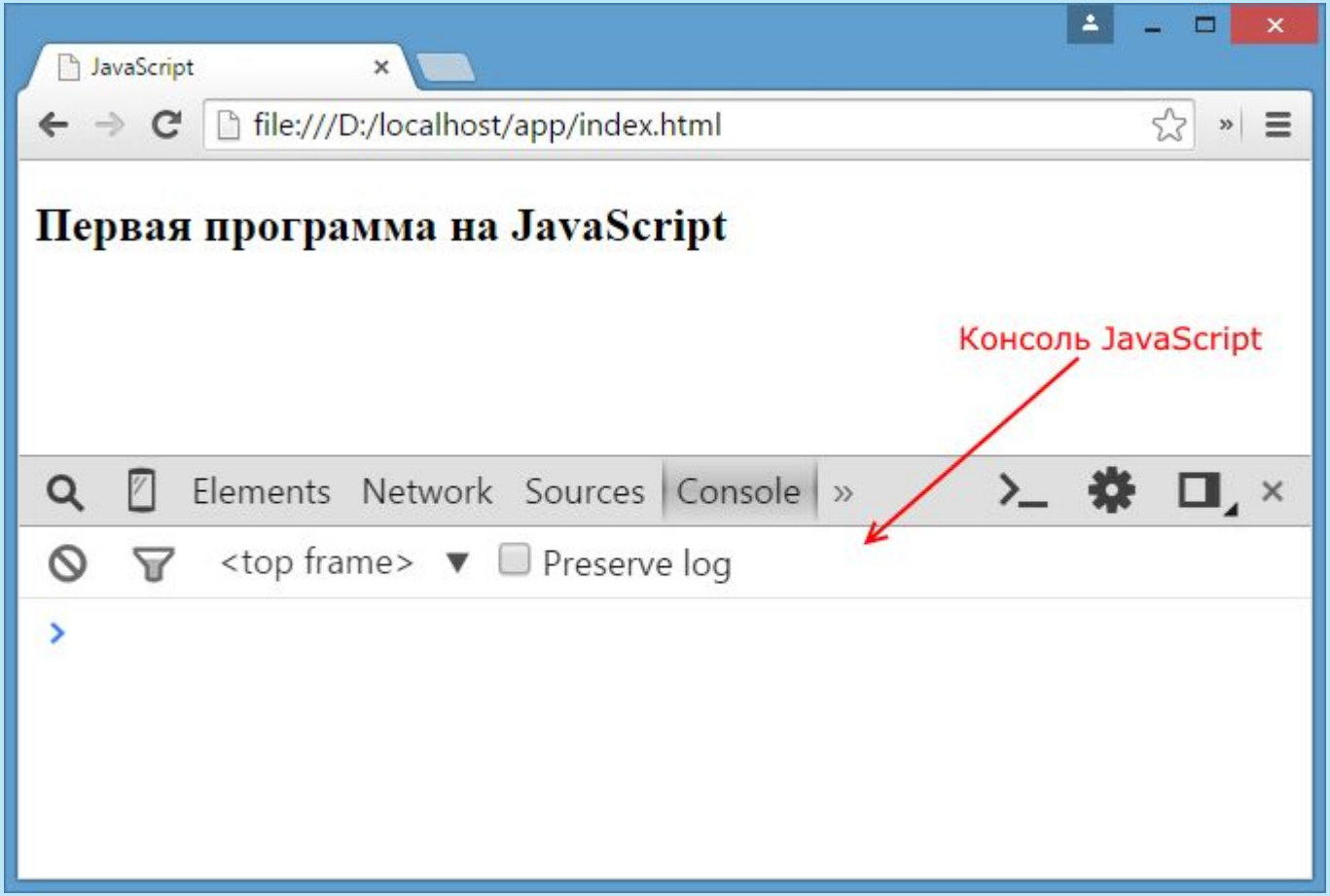
В отличие от определения кода JavaScript вынесение его во внешние файлы имеет ряд преимуществ:

- Можно повторно использовать один и тот же код на нескольких веб-страницах
- Внешние файлы JavaScript бразуер может кэшировать, за счет этого при следующем обращении к странице браузер снижает нагрузка на сервер, а браузеру надо загрузить меньший объем информации
- Код веб-страницы становится "чище". Он содержит только html-разметку, а код поведения хранится во внешних файлах. В итоге можно отделить работу по созданию кода html-страницы от написания кода JavaScript

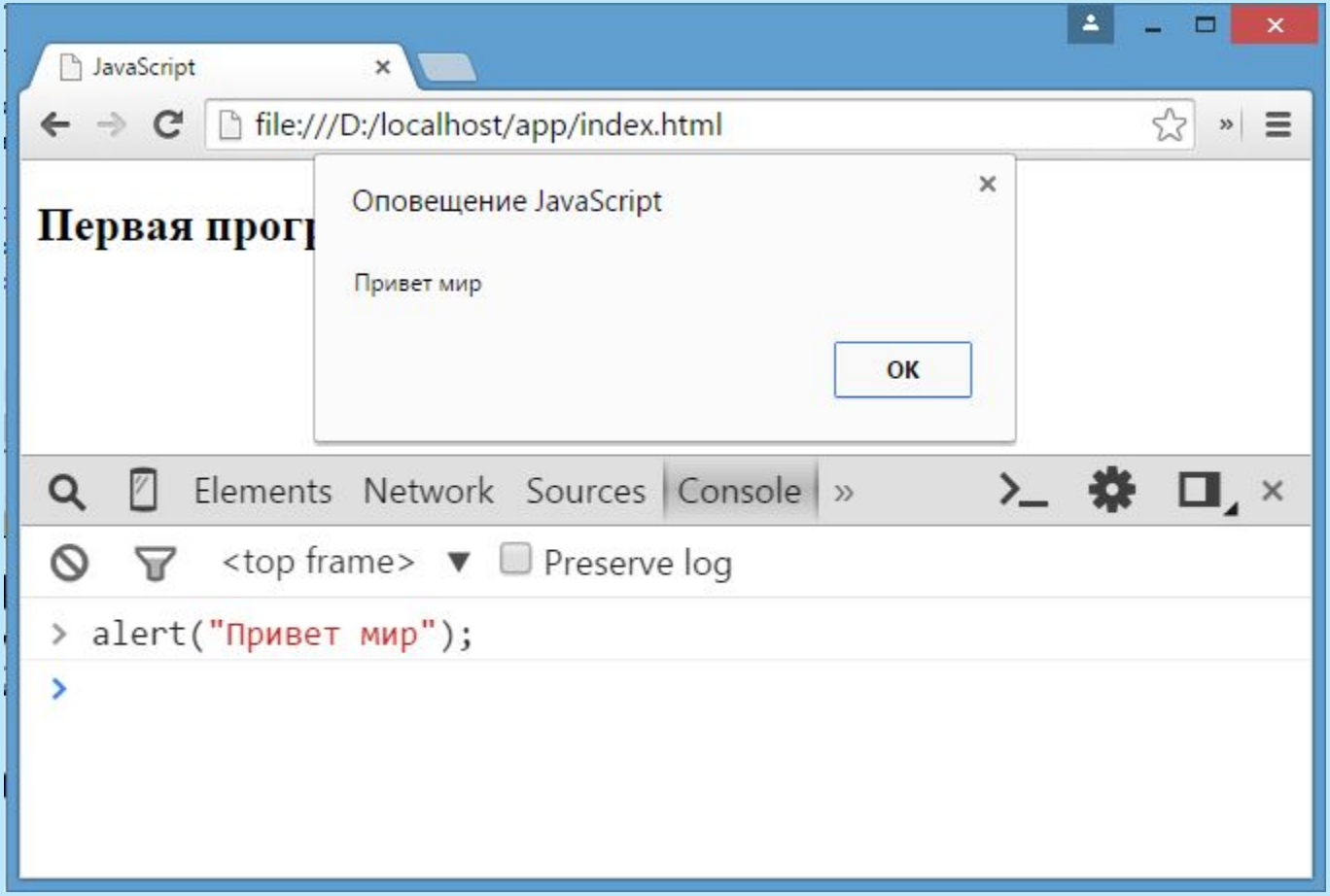
JavaScript


JavaScript
file:///D:/localhost/app/index.html
Первая программа на JavaScript
Новая вкладка Ctrl+T
Новое окно Ctrl+N
Новое окно в режиме инкогнито Ctrl+Shift+N
История Ctrl+H
Загрузки Ctrl+J
Недавние вкладки
Закладки
Масштаб - 100% +
Печать... Ctrl+P
Сохранить страницу как... Ctrl+S
Найти... Ctrl+F
Дополнительные инструменты
Изменить Вырезать Копировать Вставить
Настройки
О браузере Google Chrome
Справка
Перезапустить Chrome в режиме Windows 8
Выход Ctrl+Shift+Q
Удаление данных о просмотренных страницах... Ctrl+Shift+Del
Расширения
Диспетчер задач Shift+Esc
Добавить в панель задач...
Кодировка
Инструменты разработчика Ctrl+Shift+I
Просмотреть исходный код Ctrl+U
Консоль JavaScript Ctrl+Shift+J
Проверка устройств

JavaScript

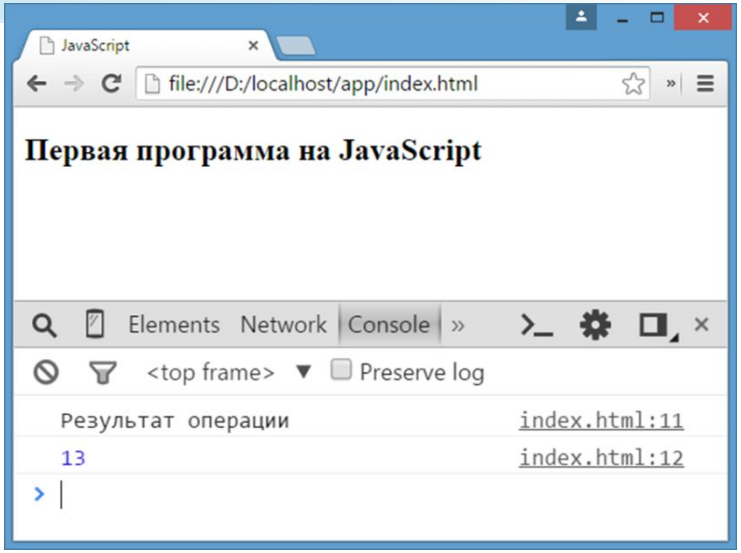

JavaScript
file:///D:/localhost/app/index.html
Первая программа на JavaScript
Консоль JavaScript
Elements
Network
Sources
Console
<top frame>
Preserve log

JavaScript


JavaScript
file:///D:/localhost/app/index.html
Первая прогр
Оповещение JavaScript
Привет мир
OK
Elements
Network
Sources
Console
<top frame>
Preserve log
> alert("Привет мир");
>

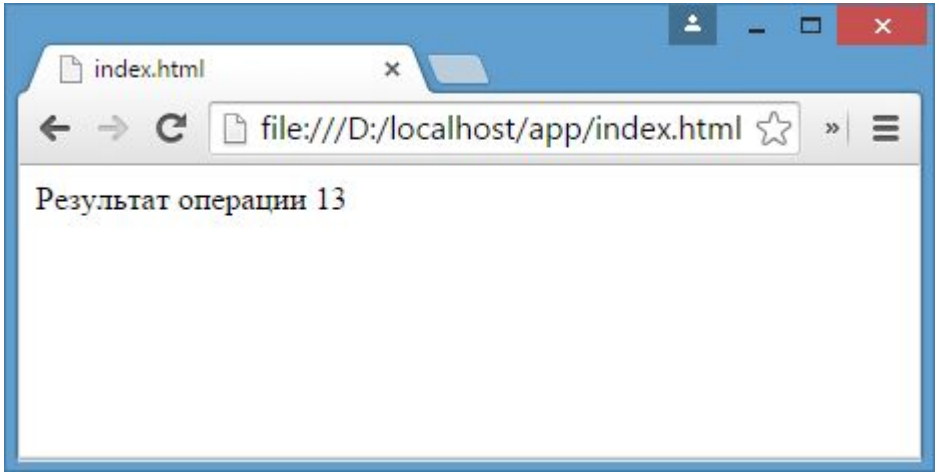
JavaScript

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
<title>JavaScript</title>
</head>
<body>
  <h2>Первая программа на
JavaScript</h2>
  <script>
  var a = 5 + 8;
  console.log("Результат операции");
  console.log(a);
  </script>
</body>
</html>
```



JavaScript

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
<title>JavaScript</title>
</head>
<body>
  <h2>Первая программа на
JavaScript</h2>
  <script>
  var a = 5 + 8;
  document.write("Результат операции ");
  document.write(a);
  </script>
</body>
</html>
```



JavaScript. Переменные и константы

Для хранения данных в программе используются **переменные**. Для создания переменных применяются ключевые слова **var** и **let**.

```
var myIncome;
// другой вариант
let myIncome2;
```

Каждая переменная имеет имя. Имя представляет собой произвольный набор алфавитно-цифровых символов, знака подчеркивания (_) или знака доллара ($), причем названия не должны начинаться с цифровых символов.

JavaScript. Переменные и константы

Список зарезервированных слов в JavaScript:

abstract, boolean, break, byte, case, catch, char, class, const, continue, debugger, default, delete, do, double, else, enum, export, extends, false, final, finally, float, for, function, goto, if, implements, import, in, instanceof, int, inteface, long, native, new, null, package, private, protected, public, return, short, static, super, switch, synchronized, this, throw, throws, transient, true, try, typeof, var, volatile, void, while, with

JavaScript. Переменные и константы

JavaScript является **регистрозависимым** языком, то есть в следующем коде объявлены две разные переменные:

```
var myIncome;
var MyIncome;
```

С помощью ключевого слова **const** можно определить **константу**, которая, как и переменная, хранит значение, однако это значение не может быть изменено.

```
const rate = 10;
```

JavaScript. Типы данных

В JavaScript имеется пять примитивных типов данных:

- String: представляет строку
- Number: представляет числовое значение
- Boolean: представляет логическое значение true или false
- undefined: указывает, что значение не установлено
- null: указывает на неопределенное значение

Все данные, которые не попадают под вышеперечисленные пять типов, относятся к типу **object**

JavaScript. Числовые данные

Числа в JavaScript могут иметь две формы:

Целые числа, например, 35. Можно использовать как положительные, так и отрицательные числа.

Дробные числа (числа с плавающей точкой), например, 3.5575. Опять же можно использовать как положительные, так и отрицательные числа.

```
var x = 45;
var y = 23.897;
```

В качестве разделителя между целой и дробной частями, как и в других языках программирования, используется точка.

JavaScript. Строки

Тип string представляет строки.

Можно использовать как двойные, так и одинарные кавычки: "Привет мир" и 'Привет мир'. Единственно ограничение: тип закрывающей кавычки должен быть тот же, что и тип открывающей, то есть либо обе двойные, либо обе одинарные.

Если внутри строки встречаются кавычки, то их нужно экранировать слешем.

```
var companyName = "Бюро \"Рога и копыта\"";
```

Также можно внутри стоки использовать другой тип кавычек:

```
var companyName1 = "Бюро 'Рога и копыта'";
var companyName2 = 'Бюро "Рога и копыта"';
```

JavaScript. Тип Boolean

Тип Boolean представляет булевые или логические значения true и false (то есть да или нет):

- var isAlive = true;
- var isDead = false;

JavaScript. null и undefined

Когда мы только определяем переменную без присвоения ей начального значения, она представляет тип undefined:

```
var isAlive;
console.log(isAlive); // выведет undefined
```

Присвоение значение null означает, что переменная имеет некоторое неопределенное значение (не число, не строка, не логическое значение), но все-таки имеет значение (undefined означает, что переменная не имеет значения):

```
var isAlive;
console.log(isAlive); // undefined
isAlive = null;
console.log(isAlive); // null
isAlive = undefined;  // снова установим тип undefined
console.log(isAlive); // undefined
```

JavaScript. Тип object

Тип object представляет сложный объект.
Простейшее определение объекта представляют фигурные скобки:

```
var user = {};
```

Объект может иметь различные свойства и методы:

```
var user = {name: "Tom", age:24};
console.log(user.name);
```

В данном случае объект называется user, и он имеет два свойства: name и age.

JavaScript. Слабая типизация

JavaScript является языком со слабой типизацией. Это значит, что переменные могут динамически менять тип. Например:

```
var xNumber;  // тип undefined
console.log(xNumber);
xNumber = 45;  // тип number
console.log(xNumber);
xNumber = "45"; // тип string
console.log(xNumber);
```

JavaScript. Слабая типизация

```
var xNumber = 45;  // тип number

var yNumber = xNumber + 5;

console.log(yNumber); // 50


xNumber = "45"; // тип string

var zNumber = xNumber + 5

console.log(zNumber); // 455
```

JavaScript. Оператор typeof

С помощью оператора **typeof** можно получить тип переменной:

```
var name = "Tom";
console.log(typeof name); // string
var income = 45.8;
console.log(typeof income); // number
var isEnabled = true;
console.log(typeof isEnabled); // boolean
var undefVariable;
console.log(typeof undefVariable); // undefined
```

JavaScript. Операции с переменными

Сложение:

```
var x = 10;
var y = x + 50;
```

Вычитание:

```
var x = 100;
var y = x - 50;
```

Умножение:

```
var x = 4;
var y = 5;
var z = x * y;
```

JavaScript. Операции с переменными

Деление:

```
var x = 40;
var y = 5;
var z = x / y;
```

Деление по модулю (оператор %) возвращает остаток от деления:

```
var x = 40;
var y = 7;
var z = x % y;
console.log(z); // 5
```

JavaScript. Операции с переменными

Инкремент:

```
var x = 5;
x++; // x = 6
```

Оператор инкремента ++ увеличивает переменную на единицу.

Существует префиксный инкремент, который сначала увеличивает переменную на единицу, а затем возвращает ее значение.

И есть постфиксный инкремент, который сначала возвращает значение переменной, а затем увеличивает его на единицу.

JavaScript. Операции с переменными

```
// префиксный инкремент
var x = 5;
var z = ++x;
console.log(x); // 6
console.log(z); // 6

// постфиксный инкремент
var a = 5;
var b = a++;
console.log(a); // 6
console.log(b); // 5
```

JavaScript. Операции с переменными

Декремент уменьшает значение переменной на единицу. Также есть префиксный и постфиксный декремент:

```
// префиксный декремент
var x = 5;
var z = --x;
console.log(x); // 4
console.log(z); // 4

// постфиксный декремент
var a = 5;
var b = a--;
console.log(a); // 4
console.log(b); // 5
```

JavaScript. Операции с переменными

Все операции выполняются слева направо и различаются по приоритетам: сначала операции инкремента и декремента, затем выполняются умножение и деление, а потом сложение и вычитание.

Чтобы изменить стандартный ход выполнения операций, часть выражений можно поместить в скобки:

```
var x = 10;
var y = 5 + (6 - 2) * --x;
console.log(y); //41
```

JavaScript. Операции с переменными

Операции присваивания =

Приравнивает переменной определенное значение: var x = 5;

Сложение с последующим присвоением результата.

```
var a = 23;
a += 5; // аналогично a = a + 5
console.log(a); // 28
```

JavaScript. Операции с переменными

Вычитание с последующим присвоением результата.

```
var a = 28;
a -= 10; // аналогично a = a - 10
console.log(a); // 18
```

Умножение с последующим присвоением результата:

```
var x = 20;
x *= 2; // аналогично x = x * 2
console.log(x); // 40
```

JavaScript. Операции с переменными

Деление с последующим присвоением результата:

```
var x = 40;
x /= 4; // аналогично x = x / 4
console.log(x); // 10
```

Получение остатка от деления с последующим присвоением результата:

```
var x = 10;
x %= 3; // аналогично x = x % 3
console.log(x); // 1, так как 10 - 3*3 = 1
```

JavaScript. Операции с переменными

Операторы сравнения

==

Оператор равенства сравнивает два значения, и если они равны, возвращает true, иначе возвращает false: x == 5

===

Оператор тождественности также сравнивает два значения и их тип, и если они равны, возвращает true, иначе возвращает false: x === 5

!=

Сравнивает два значения, и если они не равны, возвращает true, иначе возвращает false: x != 5

JavaScript. Операции с переменными

!==

Сравнивает два значения и их типы, и если они не равны, возвращает true, иначе возвращает false: x !== 5

>

Сравнивает два значения, и если первое больше второго, то возвращает true, иначе возвращает false: x > 5

<

Сравнивает два значения, и если первое меньше второго, то возвращает true, иначе возвращает false: x < 5

>=

Сравнивает два значения, и если первое больше или равно второму, то возвращает true, иначе возвращает false: x >= 5

<=

Сравнивает два значения, и если первое меньше или равно второму, то возвращает true, иначе возвращает false: x <= 5

JavaScript. Операции с переменными

Логические операции

Логические операции применяются для объединения результатов двух операций сравнения. В JavaScript есть следующие логические операции:

&&

Возвращает true, если обе операции сравнения возвращают true, иначе возвращает false:

```
var income = 100;
var percent = 10;
var result = income > 50 && percent < 12;
console.log(result); //true
```

JavaScript. Операции с переменными

||

Возвращает true, если хотя бы одна операция сравнения возвращают true, иначе возвращает false:

```
var income = 100;
var isDeposit = true;
var result = income > 50 || isDeposit == true;
console.log(result); //true
```

JavaScript. Операции с переменными

!

Возвращает true, если операция сравнения возвращает false:

```
var income = 100;
var result1 = !(income > 50);
console.log(result1); // false, так как income > 50 возвращает true

var isDeposit = false;
var result2 = !isDeposit;
console.log(result2); // true
```

JavaScript. Операции с переменными

Операции со строками

Строки могут использовать оператор + для объединения.

```
var name = "Том";
var surname = "Сойер"
var fullname = name + " " + surname;
console.log(fullname); //Том Сойер
```

Если одно из выражений представляет строку, а другое - число, то число преобразуется к строке и выполняется операция объединения строк:

```
var name = "Том";
var fullname = name + 256;
console.log(fullname); //Том256
```

JavaScript. Преобразования данных

Для преобразования строки в число применяется функция **parseInt():**

```
var number1 = "46";
var number2 = "4";
var result = parseInt(number1) + parseInt(number2);
console.log(result); // 50
```

Для преобразования строк в дробные числа применяется функция **parseFloat()**:

```
var number1 = "46.07";
var number2 = "4.98";
var result = parseFloat(number1) + parseFloat(number2);
console.log(result); //51.05
```

JavaScript. Преобразования данных

Если методу не удастся выполнить преобразование, то он возвращает значение NaN (Not a Number), которое говорит о том, что строка не представляет число и не может быть преобразована.

С помощью специальной функции isNaN() можно проверить, представляет ли строка число. Если строка не является числом, то функция возвращает true, если это число - то false:

```
var num1 = "javascript";
var num2 = "22";
var result = isNaN(num1);
console.log(result); // true - num1 не является числом

result = isNaN(num2);
console.log(result); //  false - num2 - это число
```

JavaScript. Массивы

Для работы с наборами данных предназначены массивы. Для создания массива применяется выражение new Array():

```
var myArray = new Array();
```

Существует также более короткий способ инициализации массива:

```
var myArray = [];
```

В данном случае создается пустой массив. Но можно также добавить в него начальные данные:

```
var people = ["Tom", "Alice", "Sam"];
console.log(people);
```

JavaScript. Массивы

Для обращения к отдельным элементам массива используются индексы. Отсчет начинается с нуля, то есть первый элемент будет иметь индекс 0, а последний - 2:

```
var people = ["Tom", "Alice", "Sam"];
console.log(people[0]); // Tom
var person3 = people[2]; // Sam
console.log(person3); // Sam
```

Если мы попробуем обратиться к элементу по индексу больше размера массива, то мы получим undefined:

```
var people = ["Tom", "Alice", "Sam"];
console.log(people[7]); // undefined
```

JavaScript. Массивы

В отличие от других языков, как C# или Java, можно установить элемент, который изначально не установлен:

```
var people = ["Tom", "Alice", "Sam"];
console.log(people[7]); // undefined - в массиве только три элемента

people[7] = "Bob";
console.log(people[7]); // Bob\
```

Также стоит отметить, что в отличие от ряда языков программирования в JavaScript массивы не являются строго типизированными, один массив может хранить данные разных типов:

```
var objects = ["Tom", 12, true, 3.14, false];
console.log(objects);
```

JavaScript. Массивы

spread-оператор

spread-оператор ... позволяет взять значения из массива по отдельности:

```
let numbers = [1, 2, 3, 4];
console.log(...numbers);    // 1 2 3 4
console.log(numbers);       // [1, 2, 3, 4]
```

spread-оператор указывается перед массивом. В результате выражение ...numbers возвратит набор чисел, но это будет не массив, а именно отдельные значения.

JavaScript. Массивы

Многомерные массивы

Массивы могут быть одномерными и многомерными. Каждый элемент в многомерном массиве может представлять собой отдельный массив. Выше мы рассматривали одномерный массив, теперь создадим многомерный массив:

```
var numbers1 = [0, 1, 2, 3, 4, 5 ]; // одномерный массив
var numbers2 = [[0, 1, 2], [3, 4, 5] ]; // двумерный массив
```

JavaScript. Условные конструкции

Выражение if

Конструкция if проверяет некоторое условие и если это условие верно, то выполняет некоторые действия.

Общая форма конструкции if:

if(условие) действия;

Например:

```
var income = 100;
if(income > 50) alert("доход больше 50");
```

JavaScript. Условные конструкции

Если необходимо выполнить по условию набор инструкций, то они помещаются в блок из фигурных скобок:

```
var income = 100;
if(income > 50){
    var message = "доход больше 50";
   alert(message);
}
```

JavaScript. Условные конструкции

В конструкции if также можно использовать блок else. Данный блок содержит инструкции, которые выполняются, если условие после if ложно, то есть равно false:

```
var age = 17;
if(age >= 18){
    alert("Вы допущены к программе кредитования");
}
else{
   alert("Вы не можете участвовать в программе, так как возраст меньше 18");
}
```

JavaScript. Условные конструкции

С помощью конструкции else if можно добавить альтернативное условие к блоку if:

```
var income = 300;
if(income < 200){
    alert("Доход ниже среднего");
}
else if(income >= 200 && income <= 400){
        alert("Средний доход");
}
else{
        alert("Доход выше среднего");
}
```

JavaScript. Условные конструкции

В JavaScript любая переменная может применяться в условных выражениях, но не любая переменная представляет тип boolean.

И в этой связи возникает вопрос, что возвратит та или иная переменная - true или false? Много зависит от типа данных, который представляет переменная:

- undefined. Возвращает false
- null. Возвращает false
- Boolean. Если переменная равна false, то возвращается false. Соответствено если переменная равна true, то возвращается true
- Number. Возвращает false, если число равно 0 или NaN (Not a Number), в остальных случаях возвращается true
- String. Возвращает false, если переменная равна пустой строке, то есть ее длина равна 0, в остальных случаях возвращается true
- Object. Всегда возвращает true

JavaScript. Условные конструкции

Конструкция switch..case является альтернативой использованию конструкции if..else if..else и также позволяет обработать сразу несколько условий:

```
var income = 300;
switch(income){
    case 100 :
        console.log("Доход равен 100");
        break;
    case 200 :
        console.log("Доход равен 200");
        break;
    case 300 :
        console.log("Доход равен 300");
        break;
}
```

JavaScript. Условные конструкции

Тернарная операция

Тернарная операция состоит из трех операндов и имеет следующее определение:

[первый операнд - условие] ? [второй операнд] : [третий операнд].

В зависимости от условия тернарная операция возвращает второй или третий операнд: если условие равно true, то возвращается второй операнд; если условие равно false, то третий. Например:

```
var a = 1;
var b = 2;
var result = a < b ? a + b : a - b;
console.log(result); // 3
```

Если значение переменной a меньше значения переменной b, то переменная result будет равняться a + b. Иначе значение result будет равняться a - b.

JavaScript. Циклы

Циклы позволяют в зависимости от определенных условий выполнять некоторое действие множество раз. В JavaScript имеются следующие виды циклов:

- **for**
- **for..in**
- **for..of**
- **while**
- **do..while**

JavaScript. Циклы

Цикл for

Цикл for имеет следующее формальное определение:

```
for ([инициализация счетчика]; [условие]; [изменение счетчика]){
    // действия
}
```

Например, используем цикл for для перебора элементов массива:

```
var people = ["Tom", "Alice", "Bob", "Sam"];
for(var i = 0; i<people.length; i++){
    console.log(people[i]);
}
```

JavaScript. Циклы

for..in

Цикл for..in предназначен для перебора массивов и объектов. Его формальное определение:

```
for (индекс in массив) {
    // действия
}
```

Например, переберем элементы массива:

```
var people = ["Tom", "Alice", "Bob", "Sam"];
for(var index in people){
    console.log(people[index]);
}
```

JavaScript. Циклы

Цикл for...of

Цикл for...of похож на цикл for...in и предназначен для перебора коллекций, например, массивов:

```
let users = ["Tom", "Bob", "Sam"];
for(let val of users)
    console.log(val);
```

JavaScript. Циклы

Цикл while

Цикл while выполняется до тех пор, пока некоторое условие истинно. Его формальное определение:

```
while(условие){
    // действия
}
```

Опять же выведем с помощью while элементы массива:

```
var people = ["Tom", "Alice", "Bob", "Sam"];
var index = 0;
while(index < people.length){
    console.log(people[index]);
    index++;
}
```

JavaScript. Циклы

do..while

В цикле do сначала выполняется код цикла, а потом происходит проверка условия в инструкции while. И пока это условие истинно, цикл повторяется. Например:

```
var x = 1;
do{
    console.log(x * x);
    x++;
}while(x < 10)
```

JavaScript. Циклы

Операторы continue и break

Иногда бывает необходимо выйти из цикла до его завершения. В этом случае мы можем воспользоваться оператором break:

```
var array = [ 1, 2, 3, 4, 5, 12, 17, 6, 7 ];
for (var i = 0; i < array.length; i++)
{
    if (array[i] > 10)
        break;
    document.write(array[i] + "</br>");
}
```

JavaScript. Функции

Функции представляют собой набор инструкций, выполняющих определенное действие или вычисляющих определенное значение.

Синтаксис определения функции:

```
function имя_функции([параметр [, ...]]){
    // Инструкции
}
```

JavaScript. Функции

Определение функции начинается с ключевого слова function, после которого следует имя функции.

Наименование функции подчиняется тем же правилам, что и наименование переменной: оно может содержать только цифры, буквы, символы подчеркивания и доллара ($) и должно начинаться с буквы, символа подчеркивания или доллара.

После имени функции в скобках идет перечисление параметров.

Даже если параметров у функции нет, то просто идут пустые скобки.

Затем в фигурных скобках идет тело функции, содержащее набор инструкций.

JavaScript. Функции

Необязательно давать функциям определенное имя. Можно использовать анонимные функции:

```
var display = function(){  // определение функции
    document.write("функция в JavaScript");
}
display();
```

Фактически мы определяем переменную display и присваиваем ей ссылку на функцию. А затем по имени переменной функция вызывается.

JavaScript. Функции

Параметры функции

Рассмотрим передачу параметров:

```
function display(x){  // определение функции
    var z = x * x;
    document.write(x + " в квадрате равно " + z);
}
display(5); // вызов функции
```

Функция display принимает один параметр - x. Поэтому при вызове функции мы можем передать для него значение, например, число 5, как в данном случае.

JavaScript. Функции

Если функция принимает несколько параметров, то с помощью spread-оператора ... мы можем передать набор значений для этих параметров из массива:

```
function sum(a, b, c){
    let d = a + b + c;
    console.log(d);
}
sum(1, 2, 3);
let nums = [4, 5, 6];
sum(...nums);
```

Во втором случае в функцию передается числа из массива nums. Но чтобы передавался не просто массив, как одно значение, а именно числа из этого массива, применяется spread-оператор (многоточие ...).

JavaScript. Функции

Необязательные параметры

Функция может принимать множество параметров, но при этом часть или все параметры могут быть необязательными. Если для параметров не передается значение, то по умолчанию они имеют значение "undefined".

```
function display(x, y){
    if(y === undefined) y = 5;
    if(x === undefined) x = 8;
    let z = x * y;
    console.log(z);
}
display();  // 40
display(6); // 30
display(6, 4) // 24
```

JavaScript. Функции

При необходимости можно получить все переданные параметры через глобально доступный массив arguments:

```
function display(){
    var z = 1;
    for(var i=0; i<arguments.length; i++)
        z *= arguments[i];
    console.log(z);
}
display(6); // 6
display(6, 4) // 24
display(6, 4, 5) // 120
```

При этом даже не важно, что при определении функции мы не указали никаких параметров, мы все равно можем их передать и получить их значения через массив arguments.

JavaScript. Функции

Неопределенное количество параметров

С помощью spread-оператора можно указать, что с помощью параметра можно передать переменное количество значений:

```
function display(season, ...temps){
    console.log(season);
    for(index in temps){
        console.log(temps[index]);
    }
}
display("Весна", -2, -3, 4, 2, 5);
display("Лето", 20, 23, 31);
```

В данном случае второй параметр ...temps указывает, что вместо него можно передать разное количество значений. В самой функции temps фактически представляет массив переданных значений, которые мы можем получить. При этом несмотря на это, при вызове функции в нее передается не массив, а именно отдельные значения.

JavaScript. Функции

Результат функции

Функция может возвращать результат. Для этого используется оператор return:

```
var y = 5;
var z = square(y);
document.write(y + " в квадрате равно " + z);
function square(x) {
  return x * x;
}
```

После оператора return идет значение, которое надо возвратить из метода. В данном случае это квадрат числа x.

JavaScript. Область видимости переменных

Глобальные переменные

Все переменные, которые объявлены вне функций, являются глобальными:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
</head>
<body>
<script>
var x = 5;
let d = 8;
function displaySquare(){
     var z = x * x;
    console.log(z);}
</script>
</body>
</html>
```

JavaScript. Область видимости переменных

Сокрытие переменных

Что если у нас есть две переменных - одна глобальная, а другая локальная, которые имеют одинаковое имя:

```
var z = 89;
function displaySquare(){
    var z = 10;
    console.log(z); // 10
}
displaySquare(); // 10
```

В этом случае в функции будет использоваться та переменная z, которая определена непосредственно в функции. То есть локальная переменная скроет глобальную.

JavaScript. Область видимости переменных

Необъявленные переменные

Если мы не используем ключевое слово при определении переменной в функции, то такая переменная будет глобальной. Например:

```
function bar(){
    foo = "25";
}
bar();
console.log(foo);   // 25
```

JavaScript. Область видимости переменных

strict mode

Определение глобальных переменных в функциях может вести к потенциальным ошибкам. Чтобы их избежать используется строгий режим или strict mode:

```
"use strict";
function bar(){
    foo = "25";
}
bar();
console.log(foo);
```

В этом случае мы получим ошибку SyntaxError: Unexpected identifier, которая говорит о том, что переменная foo не определена.

JavaScript. Область видимости переменных

Установить режим strict mode можно двумя способами:

- добавить выражение "use strict" в начало кода JavaScript, тогда strict mode будет применяться для всего кода
- добавить выражение "use strict" в начало тела функции, тогда strict mode будет применяться только для этой функции

JavaScript. Объекты

Есть несколько способов создания нового объекта.

Первый способ заключается в использовании конструктора Object:

```
var user = new Object();
```

В данном случае объект называется user. Он определяется также, как и любая обычная переменная с помощью ключевого слова var.

Выражение new Object() представляет вызов конструктора - функции, создающей новый объект. Для вызова конструктора применяется оператор new. Вызов конструктора фактически напоминает вызов обычной функции.

Второй способ создания объекта представляет использование фигурных скобок:

```
var user = {};
```

JavaScript. Объекты

Свойства объекта

После создания объекта можно определить в нем свойства. Чтобы определить свойство, надо после названия объекта через точку указать имя свойства и присвоить ему значение:

```
var user = {};
user.name = "Tom";
user.age = 26;
```

JavaScript. Объекты

Также можно определить свойства при определении объекта:

```
var user = {
    name: "Tom",
    age: 26
};
```

Кроме того, доступен сокращенный способ определения свойств:

```
var name = "Tom";
var age = 34;
var user = {name, age};
console.log(user.name);     // Tom
console.log(user.age);      // 34
```

JavaScript. Объекты

Методы объекта

Определяют его поведение или действия, которые он производит. Методы представляют собой функции. Например, определим метод, который бы выводил имя и возраст человека:

```
var user = {};
user.name = "Tom";
user.age = 26;
user.display = function(){
    console.log(user.name);
    console.log(user.age);
};
 // вызов метода
user.display();
```

JavaScript. Объекты

Также методы могут определяться непосредственно при определении объекта:

```
var user = {
    name: "Tom",
    age: 26,
    display: function(){
        console.log(this.name);
        console.log(this.age);
    }
};
```

JavaScript. Объекты

Чтобы обратиться к свойствам или методам объекта внутри этого объекта, используется ключевое слово **this**.

Оно означает ссылку на текущий объект.

Также можно использовать сокращенный способ определения методов, когда двоеточие и слово function опускаются.

JavaScript. Объекты

```
var user = {
    name: "Tom",
    age: 26,
    display(){
       console.log(this.name, this.age);
    },
    move(place){
        console.log(this.name, "goes to", place);
    }
};
user.display(); // Tom 26
user.move("the shop");  // Tom goes to the shop
```

JavaScript. Объекты

Существует также альтернативный способ определения свойств и методов с помощью синтаксиса массивов:

```
var user = {};
user["name"] = "Tom";
user["age"] = 26;
user["display"] = function(){
    console.log(user.name);
    console.log(user.age);
};
// вызов метода
user["display"]();
```

JavaScript. Объекты

Названия свойств и методов объекта всегда представляют строки. Предыдущее определение объекта можно переписать:

```
var user = {
    "name": "Tom",
    "age": 26,
    "display": function(){
        console.log(user.name);
        console.log(user.age);
    }
};
// вызов метода
user.display();
```

JavaScript. Объекты

Удаление свойств

Можно также удалять свойства и методы с помощью оператора **delete**.

И как и в случае с добавлением можно удалять свойства двумя способами.

- Первый способ - использование нотации точки:

  delete объект.свойство

- Либо использовать синтаксис массивов:

  delete объект["свойство"]

JavaScript. Объекты

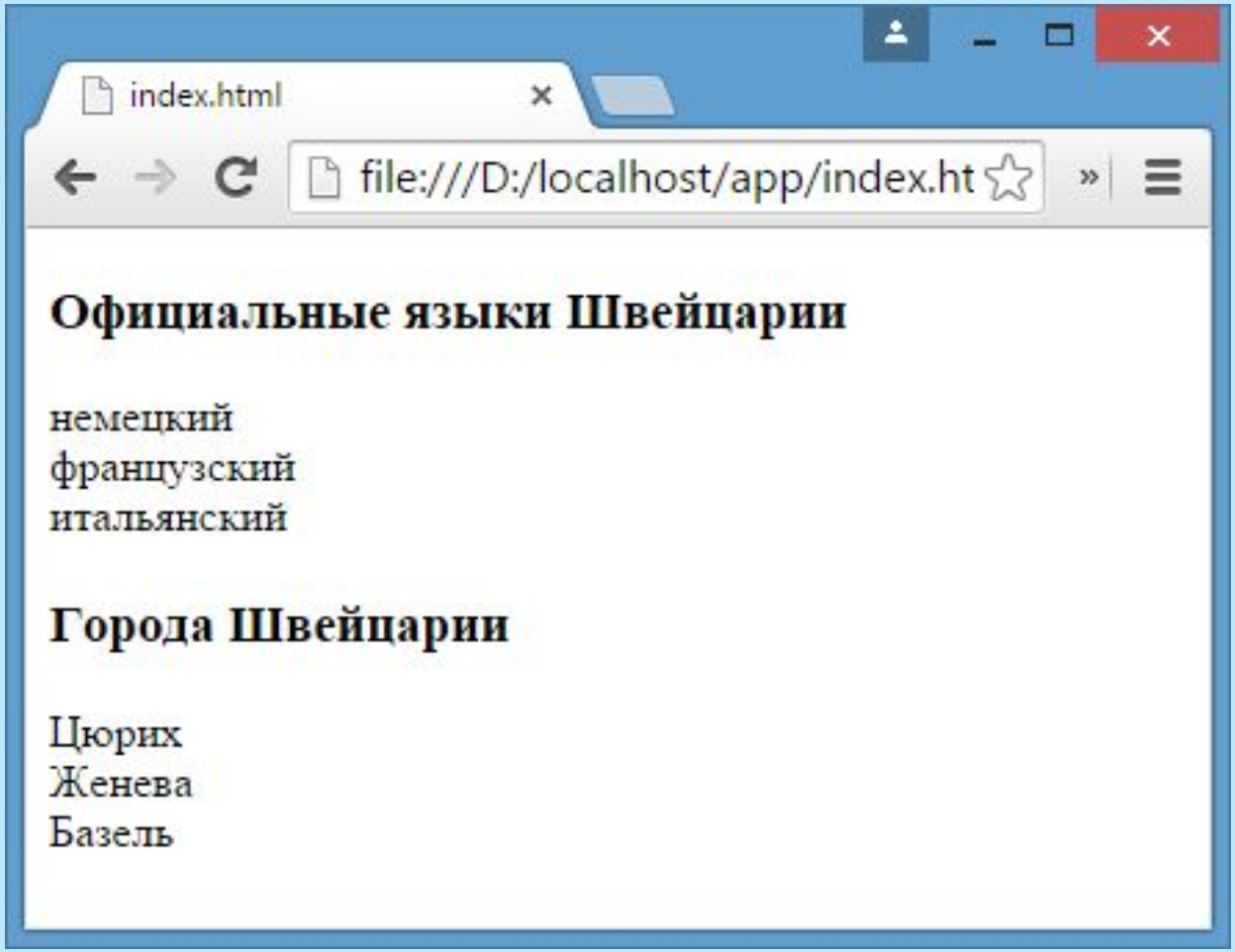
Одни объекты могут содержать в качестве свойств другие объекты.

```
var country = {
    name: "Германия",
    language: "немецкий",
    capital:{
        name: "Берлин",
        population: 3375000,
        year: 1237
    }
};
console.log("Столица: " + country.capital.name); // Берлин
console.log("Население: " + country["capital"]["population"]); // 3375000
console.log("Год основания: " + country.capital["year"]); // 1237
```

JavaScript. Объекты

```
var country = {
  name: "Швейцария",
  languages: ["немецкий", "французский", "итальянский"],
  capital:{
    name: "Берн",
    population: 126598
  },
  cities: [
    { name: "Цюрих", population: 378884},
    { name: "Женева", population: 188634},
    { name: "Базель", population: 164937}
  ]
};
 // вывод всех элементов из country.languages
document.write("<h3>Официальные языки
Швейцарии</h3>");
for(var i=0; i < country.languages.length; i++)
  document.write(country.languages[i] + "<br/>");
 // вывод всех элементов из country.cities
document.write("<h3>Города Швейцарии</h3>");
for(var i=0; i < country.cities.length; i++)
  document.write(country.cities[i].name + "<br/>");
```

JavaScript. Объекты



JavaScript. Объекты

При динамическом определении в объекте новых свойств и методов перед их использованием бывает важно проверить, а есть ли уже такие методы и свойства. Для этого в JavaScript может использоваться **оператор in**:

```
var user = {};
user.name = "Tom";
user.age = 26;
user.display = function(){
    console.log(user.name);
    console.log(user.age);
};
var hasNameProp = "name" in user;
console.log(hasNameProp); // true - свойство name есть в user
var hasWeightProp = "weight" in user;
console.log(hasWeightProp); // false - в user нет свойства или метода под названием weight
```

JavaScript. Объекты

Альтернативный способ заключается на значении undefined. Если свойство или метод равен undefined, то эти свойство или метод не определены:

```
var hasNameProp = user.name!==undefined;
console.log(hasNameProp); // true
var hasWeightProp = user.weight!==undefined;
console.log(hasWeightProp); // false
```

JavaScript. Объекты

Кроме создания новых объектов JavaScript предоставляет возможность создавать новые типы объектов с помощью конструкторов. Так, одним из способов создания объекта является применение конструктора типа Object:

```
var tom = new Object();
```

После создания переменной tom она будет вести себя как объект типа Object.

Конструктор позволяет определить новый тип объекта. Тип представляет собой абстрактное описание или шаблон объекта.

JavaScript. Объекты

Определение типа может состоять из функции конструктора, методов и свойств.

Для начала определим конструктор:

```
function User(pName, pAge) {
    this.name = pName;
    this.age = pAge;
    this.displayInfo = function(){
        document.write("Имя: " + this.name + "; возраст: " + this.age + "<br/>");
    };
}
```

Конструктор - это обычная функция за тем исключением, что в ней можно установить свойства и методы. Для установки свойств и методов используется ключевое слово this:

```
this.name = pName;
```

JavaScript. Объекты

После этого в программе мы можем определить объект типа User и использовать его свойства и методы:

```
var tom = new User("Том", 26);
console.log(tom.name); // Том
tom.displayInfo();
```

Чтобы вызвать конструктор, то есть создать объект типа User, надо использовать ключевое слово new.

JavaScript. Объекты

Оператор instanceof позволяет проверить, с помощью какого конструктора создан объект. Если объект создан с помощью определенного конструктора, то оператор возвращает true:

```
var tom = new User("Том", 26);

var isUser = tom instanceof User;
var isCar = tom instanceof Car;
console.log(isUser);    // true
console.log(isCar);     // false
```

JavaScript. Объекты

Кроме непосредственного определения свойств и методов в конструкторе мы также можем использовать свойство **prototype**.

Каждая функция имеет свойство prototype, представляющее прототип функции. То есть свойство User.prototype представляет прототип объектов User.

И любые свойства и методы, которые будут определены в User.prototype, будут общими для всех объектов User.

JavaScript. Объекты

```
User.prototype.maxAge = 110;

var tom = new User("Том", 26);

var john = new User("Джон", 28);

tom.maxAge = 99;

console.log(tom.maxAge); // 99

console.log(john.maxAge); // 110
```

При обращении к свойству maxAge javascript сначала ищет это свойство среди свойств объекта, и если оно не было найдено, тогда обращается к свойствам прототипа.

JavaScript. Объекты

В JavaScript функция тоже является объектом - объектом Function и тоже имеет прототип, свойства, методы. Все функции, которые используются в программе, являются объектами Function и имеют все его свойства и методы.

Например, мы можем создать функцию с помощью конструктора Function:

```
var square = new Function('n', 'return n * n;');
console.log(square(5));
```

JavaScript. Объекты

Среди свойств объекта Function можно выделить следующие:

- **arguments**: массив аргументов, передаваемых в функцию
- **length**: определяет количество аргументов, которые ожидает функция
- **caller**: определяет функцию, вызвавшую текущую выполняющуюся функцию
- **name**: имя функции
- **prototype**: прототип функции

JavaScript. Объекты

С помощью прототипа можно определить дополнительные свойства:

```
function display(){
    console.log("привет мир");
}
Function.prototype.program ="Hello";

console.log(display.program); // Hello
```

JavaScript. Объекты

Метод call() вызывает функцию с указанным значением **this** и аргументами:

```
function add(x, y){
    return x + y;
}
var result = add.call(this, 3, 8);
console.log(result); // 11
```

this указывает на объект, для которого вызывается функция - в данном случае это глобальный объект window. После this передаются значения для параметров.

JavaScript. Объекты

На метод call() похож **метод apply()**, который также вызывает функцию и в качестве первого параметра также получает объект, для которого функция вызывается.

Только теперь в качестве второго параметра передается массив аргументов:

```
function add(x, y){
    return x + y;
}
var result = add.apply(null, [3, 8]);
console.log(result); // 11
```

JavaScript. Объекты

Поведение ключевого слова this зависит от контекста, в котором оно используется, и от того, в каком режиме оно используется - строгом или нестрогом.

В глобальном контексте this ссылается на глобальный объект. В данном случае поведение не зависит от режима (строгий или нестрогий):

```
this.alert("global alert");
this.console.log("global console");
var currentDocument = this.document;
```

JavaScript. Объекты

Контекст функции

Если скрипт запускается в строгом режиме (директива "use strict"), то **this** ссылается непосредственно на контекст функции. Иначе **this** ссылается на внешний контекст.

Например:

```
function foo(){
    var bar = "bar2";
    console.log(this.bar);
}
var bar = "bar1";
foo();  // bar1
```

JavaScript. Объекты

Контекст объекта

В контексте объекта, в том числе в его методах, ключевое слово **this** ссылается на этот же объект:

```
var o = {
    bar: "bar3",
    foo: function(){
        console.log(this.bar);
    }
}
var bar = "bar1";
o.foo();    // bar3
```

JavaScript. Объекты

С внедрением стандарта ES2015 (ES6) в JavaScript появился новый способ определения объектов - с помощью классов. Класс представляет описание объекта, его состояния и поведения, а объект является конкретным воплощением или экземпляром класса.

Для определения класса используется ключевое слово class:

```
class Person{
}
```

После слова class идет название класса (в данном случае класс называется Person), и затем в фигурных скобках определяется тело класса.

JavaScript. Объекты

Можно определить анонимный класс и присвоить его переменной:

```
let Person = class{}
```

После этого мы можем создать объекты класса с помощью конструктора:

```
class Person{}

let tom = new Person();
let bob = new Person();
```

JavaScript. Объекты

Можно определить в классе свои конструкторы.

```
class Person{
    constructor(name, age){
        this.name = name;
        this.age = age;
    }
    display(){
        console.log(this.name, this.age);
    }
}
let tom = new Person("Tom", 34);
tom.display();          // Tom 34
console.log(tom.name);  // Tom
```

JavaScript. Объекты

Конструктор определяется с помощью метода с именем **constructor**. По сути это обычный метод, который может принимать параметры. Основная цель конструктора - инициализировать объект начальными данными. И в данном случае в конструктор передаются два значения - для имени и возраста пользователя.

Для хранения состояния в классе определяются свойства. Для их определения используется ключевое слово **this**. В данном случае в классе два свойства: name и age.

JavaScript. Встроенные объекты

Объект Date позволяет работать с датами и временем в JavaScript.

Существуют различные способы создания объекта Date.

Первый способ заключается в использовании пустого конструктора без параметров:

```
var currentDate = new Date();

document.write(currentDate);
```

JavaScript. Встроенные объекты

Второй способ заключается в передаче в конструктор Date количества миллисекунд, которые прошли с начала эпохи Unix, то есть с 1 января 1970 года 00:00:00 GMT:

```
var myDate = new Date(1359270000000);
document.write(myDate); // Sun Jan 27 2013 10:00:00 GMT+0300 (RTZ 2 (зима))
```

JavaScript. Встроенные объекты

Третий способ состоит в передаче в конструктор Date дня, месяца и года:

```
var myDate = new Date("27 March 2008");
// или так
// var myDate = new Date("3/27/2008");
document.write(myDate); // Thu Mar 27 2008 00:00:00 GMT+0300 (RTZ 2 (зима))
```

JavaScript. Получение даты и времени

- **getDate()**: возвращает день месяца
- **getDay()**: возвращает день недели (отсчет начинается с 0 - воскресенье, и последний день - 6 - суббота)
- **getMonth()**: возвращает номер месяца (отсчет начинается с нуля, то есть месяц с номер 0 - январь)
- **getFullYear()**: возвращает год
- **toDateString()**: возвращает полную дату в виде строки
- **getHours()**: возвращает час (от 0 до 23)
- **getMinutes()**: возвращает минуты (от 0 до 59)
- **getSeconds()**: возвращает секунды (от 0 до 59)
- **getMilliseconds()**: возвращает миллисекунды (от 0 до 999)
- **toTimeString()**: возвращает полное время в виде строки

JavaScript. Установка даты и времени

- **setDate()**: установка дня в дате
- **setMonth()**: уставовка месяца (отсчет начинается с нуля, то есть месяц с номер 0 - январь)
- **setFullYear()**: устанавливает год
- **setHours()**: установка часа
- **setMinutes()**: установка минут
- **setSeconds()**: установка секунд
- **setMilliseconds()**: установка миллисекунд

JavaScript. Объект Math

abs()

Функция abs() возвращает абсолютное значение числа:

```
var x = -25;
document.write(Math.abs(x)); // 25
```

min() и max()

Функции min() и max() возвращают соответственно минимальное и максимальное значение из набора чисел:

```
var min = Math.min(33, 24); // 24
```

Эти функции необязательно должны принимать два числа, в них можно передавать и большее количество чисел:

ceil()

Функция ceil() округляет число до следующего наибольшего целого числа:

```
var x = Math.ceil(9.2); // 10
```

JavaScript. Объект Math

floor()

Функция floor() округляет число до следующего наименьшего целого числа:

```
var x = Math.floor(9.2); // 9
```

round()

Функция round() округляет число до следующего наименьшего целого числа, если его десятичная часть меньше 0.5. Если же десятичная часть равна или больше 0.5, то округление идет до ближайшего наибольшего целого числа:

```
var x = Math.round(5.5); // 6
```

random()

Функция random() возвращает случайное число с плавающей точкой их диапазона от 0 до 1:

```
var x = Math.random();
```

JavaScript. Объект Math

pow()

Функция pow() возвращает число в определенной степени. Например, возведем число 2 в степень 3:

```
var x = Math.pow(2, 3); // 8
```

sqrt()

```
var x = Math.sqrt(121); // 11
```

log()

Функция log() возвращает натуральный логарифм числа:

```
var x = Math.log(1); // 0
```

JavaScript. Объект Math

Константы

Кроме методов объект Math также определяет набор встроенных констант, которые можно использовать в различных вычислениях:

Math.PI (число PI): 3.141592653589793

Math.SQRT2 (квадратный корень из двух): 1.414213562373095 1

Math.SQRT1_2 (половина от квадратного корня из двух): 0.7071067811865476

Math.E (число e или число Эйлера): 2.718281828459045

Math.LN2 (натуральный логарифм числа 2): 0.6931471805599453

Math.LN10 (натуральный логарифм числа 10): 2.302585092994046

Math.LOG2E (двоичный логарифм числа e): 1.4426950408889634

Math.LOG10E (десятичный логарифм числа e): 0.4342944819032518

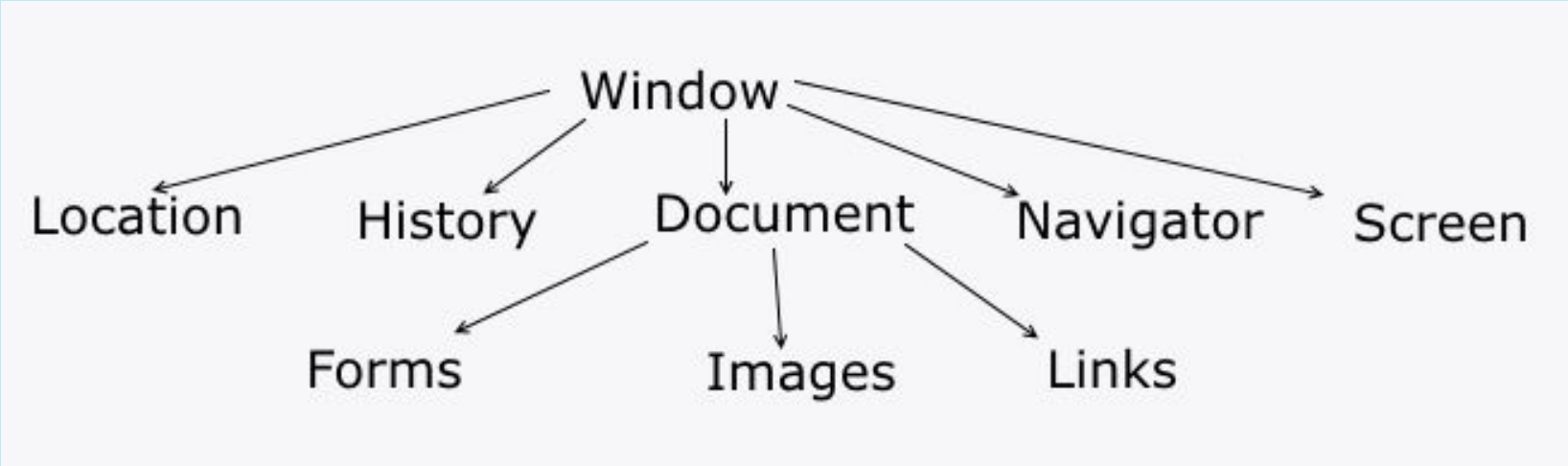
JavaScript. Работа с браузером

Большое значение в JavaScript имеет работа с веб-браузером и теми объектами, которые он предоставляет.

Например, использование объектов браузера позволяет манипулировать элементами html, которые имеются на странице, или взаимодействовать с пользователем.

Все объекты, через которые JavaScript взаимодействует с браузером, описываются таким понятием как **Browser Object Model** (Объектная Модель Браузера).

JavaScript. Работа с браузером



JavaScript. Работа с браузером

Объект window представляет собой окно веб-браузера, в котором размещаются веб-страницы. window является глобальным объектом, поэтому при доступе к его свойствам и методам необязательно использовать его имя.

Например, window имеет метод alert(), который отображает окно сообщения.

Но нам необязательно писать:

window.alert("Привет мир!");

window можно не использовать:

alert("Привет мир!");

JavaScript. Работа с браузером

Метод alert() выводит окно с сообщением:

```
alert("hello world");
```

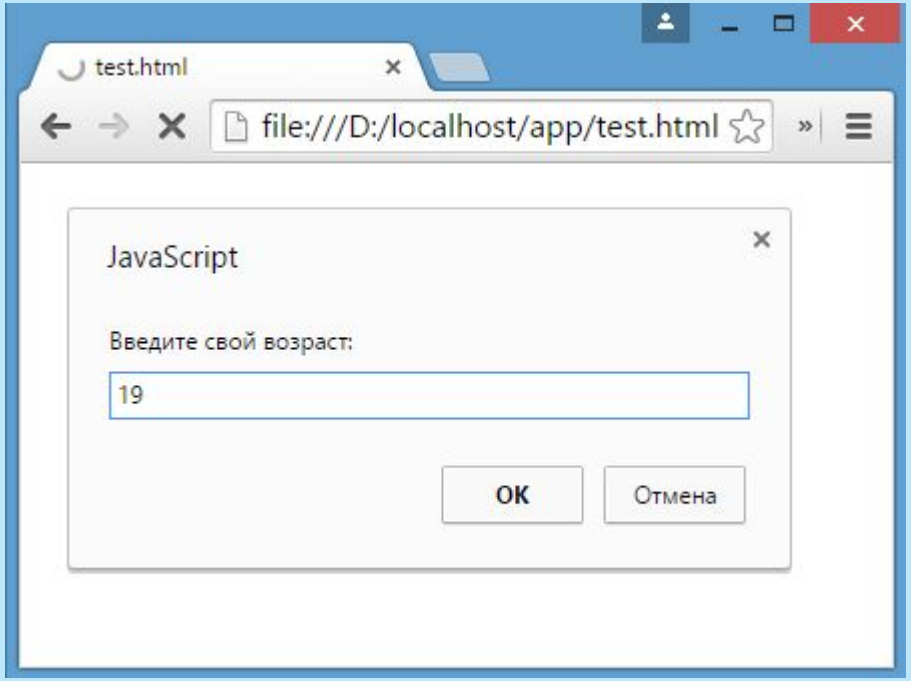
Метод confirm() отображает окно с сообщением, в котором пользователь должен подтвердить действие двух кнопок OK и Отмена. В зависимости от выбора пользователя метод возвращает true (если пользователь нажал OK) или false (если пользователь нажал кнопку Отмены):

```
var result = confirm("Завершить выполнение программы?");
if(result===true)
    document.write("Работа программы завершена");
else
    document.write("Программа продолжает работать");
```

JavaScript. Работа с браузером

Метод prompt() позволяет с помощью диалогового окна запрашивать у пользователя какие-либо данные. Данный метод возвращает введенное пользователем значение:

```
var age = prompt("Введите свой возраст:");
document.write("Вам " + age + " лет");
```



JavaScript. Работа с браузером

Объект window также предоставляет ряд методов для управления окнами браузера. Так, метод **open()** открывает определенный ресурс в новом окне браузера:

```
var popup = window.open('https://microsoft.com', 'Microsoft', 'width=400, height=400, resizable=yes');
```

Метод **open()** принимает ряд параметров: путь к ресурсу, описательное название для окна и в качестве третьего параметра набор стилевых значений окна. Метод возвращает ссылку на объект нового окна.

JavaScript. Работа с браузером

width: ширина окна в пикселях. Например, width=640

height: высота окна в пикселях. Например, height=480

left: координата X относительно начала экрана в пикселях. Например, left=0

top: координата Y относительно начала экрана в пикселях. Например, top=0

titlebar: будет ли окно иметь строку с заголовком. Например, titlebar=no

menubar: будет ли окно иметь панель меню. Например, menubar=yes

toolbar: будет ли окно иметь панели инструментов. Например, toolbar=yes

location: будет ли окно иметь адресную строку. Например, location=no

scrollbars: допускается ли наличие полос прокрутки. Например, scrollbars=yes

status: наличие статусной строки. Например, status=yes

resizable: может ли окно изменять размеры. Например, resizable=no

JavaScript. Работа с браузером

С помощью метода **close()** можно закрыть окно. Например, откроем новое окно и через 10 секунд закроем его:

```
var popup = window.open('https://microsoft.com', 'Microsoft', 'width=400,
height=400, resizable=yes');
function closeWindow(){
    popup.close();
}
setTimeout(closeWindow, 10000);
```

JavaScript. Работа с браузером

Метод **moveTo()** позволяет переместить окно на новую позицию:

```
var popup = window.open('https://microsoft.com', 'Microsoft', 'width=400,
height=400, resizable=yes');
popup.moveTo(50,50);
```

В данном случае окно перемещается на позицию с координатами x=50, y=50 относительно левого верхнего угла экрана.

Метод **resizeTo()** позволяет изменить размеры окна:

```
var popup = window.open('https://microsoft.com', 'Microsoft', 'width=400,
height=400, resizable=yes');
popup.resizeTo(500,350); // 500 - ширина и 350 - высота
```

JavaScript. Таймеры

Для выполнения действий через определенные промежутки времени в объекте window предусмотрены функции таймеров.

Есть два типа таймеров:

одни выполняются только один раз,

а другие постоянно через промежуток времени.

JavaScript. Таймеры

Функция setTimeout

Для одноразового выполнения действий через промежуток времени предназначена функция setTimeout(). Она может принимать два параметра:

```
var timerId = setTimeout(someFunction, period)
```

Параметр period указывает на промежуток, через который будет выполняться функция из параметра someFunction. А в качестве результата функция возвращает id таймера.

```
function timerFunction() {
    document.write("выполнение функции setTimeout");
}
setTimeout(timerFunction, 3000);
```

JavaScript. Таймеры

Для остановки таймера применяется функция **clearTimeout()**.

```
function timerFunction() {
    document.write("выполнение функции setTimeout");
}
var timerId = setTimeout(timerFunction, 3000);
clearTimeout(timerId);
```

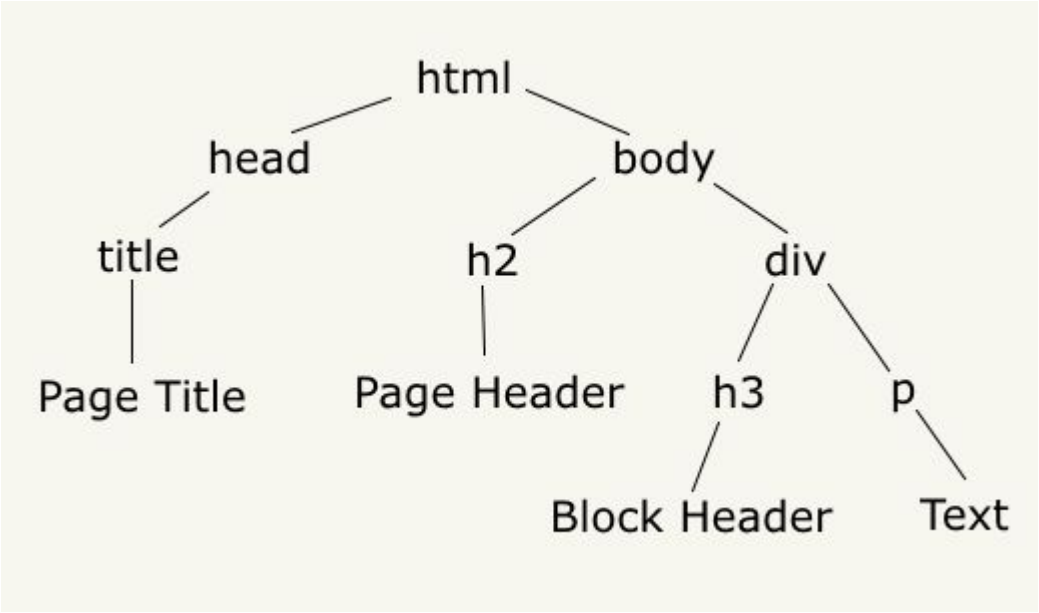
JavaScript. Таймеры

Функция setInterval

Функции setInterval() и clearInterval() работают аналогично функциям setTimeout() и clearTimeout() с той лишь разницей, что setInterval() постоянно выполняет определенную функцию через промежуток времени.

JavaScript. Работа с DOM

```
<!DOCTYPE html>
<html>
<head>
  <title>Page Title</title>
</head>
<body>
  <h2>Page Header</h2>
  <div>
    <h3>Block Header</h3>
    <p>Text</p>
  </div>
</body>
</html>
```



JavaScript. Работа с DOM

Существует следующие виды узлов:

- **Element**: html-элемент
- **Attr**: атрибут html-элемента
- **Document**: корневой узел html-документа
- **DocumentType**: DTD или тип схемы XML-документа
- **DocumentFragment**: место для временного хранения частей документа
- **EntityReference**: ссылка на сущность XML-документа
- **ProcessingInstruction**: инструкция обработки веб-страницы
- **Comment**: элемент комментария
- **Text**: текст элемента
- **CDATASection**: секция CDATA в документе XML
- **Entity**: необработанная сущность DTD
- **Notation**: нотация, объявленная в DTD

JavaScript. Работа с DOM

Для поиска элементов на странице применяются следующие методы:

- getElementById(value): выбирает элемент, у которого атрибут id равен value
- getElementsByTagName(value): выбирает все элементы, у которых тег равен value
- getElementsByClassName(value): выбирает все элементы, которые имеют класс value
- querySelector(value): выбирает первый элемент, который соответствует css-селектору value
- querySelectorAll(value): выбирает все элементы, которые соответствуют css-селектору value

JavaScript. Работа с DOM

Объект document позволяет обратиться к определенным элементам веб-страницы через свойства:

- documentElement: предоставляет доступ к корневому элементу <html>
- body: предоставляет доступ к элементу <body> на веб-странице
- images: содержит коллекцию всех объектов изображений (элементов img)
- links: содержит коллекцию ссылок - элементов <a> и <area>, у которых определен атрибут href
- anchors: предоставляет доступ к коллекции элементов <a>, у которых определен атрибут name
- forms: содержит коллекцию всех форм на веб-странице

JavaScript. Работа с DOM

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
</head>
<body>
  <img src="picture1.png" alt="Картинка 1" />
  <img src="picture2.png" alt="Картинка 2" />
  <img src="picture3.png" alt="Картинка 3" />
<script>
var images = document.images;
// изменим первое изображение
images[0].src="pics/picture_4.jpg";
images[0].alt="Новая картинка";
// перебирем все изображения
for(var i=0; i<images.length;i++){

  document.write("<br/>" + images[i].src);
  document.write("<br/>" + images[i].alt);
}
</script>
</body>
</html>
```

JavaScript. Работа с DOM

Каждый отдельный узел, будь то html-элемент, его атрибут или текст, в структуре DOM представлен объектом **Node**. Этот объект предоставляет ряд свойств:

- **childNodes**: содержит коллекцию дочерних узлов
- **firstChild**: возвращает первый дочерний узел текущего узла
- **lastChild**: возвращает последний дочерний узел текущего узла
- **previousSibling**: возвращает предыдущий элемент, который находится на одном уровне с текущим
- **nextSibling**: возвращает следующий элемент, который находится на одном уровне с текущим
- **ownerDocument**: возвращает корневой узел документа
- **parentNode**: возвращает элемент, который содержит текущий узел
- **nodeName**: возвращает имя узла
- **nodeType**: возвращает тип узла в виде числа
- **nodeValue**: возвращает или устанавливает значение узла в виде простого текста

JavaScript. События

Для взаимодействия с пользователем в JavaScript определен механизм событий. Например, когда пользователь нажимает кнопку, то возникает событие нажатия кнопки. В коде JavaScript мы можем определить возникновение события и как-то его обработать.

В JavaScript есть следующие типы событий:

- События мыши (перемещение курсора, нажатие мыши и т.д.)
- События клавиатуры (нажатие или отпускание клавиши клавиатуры)
- События жизненного цикла элементов (например, событие загрузки веб-станицы)
- События элементов форм (нажатие кнопки на форме, выбор элемента в выпадающем списке и т.д.)
- События, возникающие при изменении элементов DOM
- События, возникающие при касании на сенсорных экранах
- События, возникающие при возникновении ошибок

JavaScript. События

Рассмотрим простейшую обработку событий. Например, на веб-странице у нас есть следующий элемент div:

```
<div id="rect" onclick="alert('Нажато')"
style="width:50px;height:50px;background-color:blue;"></div>
```

Здесь определен обычный блок div, который имеет атрибут onclick, который задает обработчик события нажатия на блок div. То есть, чтобы обработать какое-либо событие, нам надо определить для него обработчик. Обработчик представляет собой код на языке JavaScript. В данном случае обработчик выглядит довольно просто:

```
alert('Нажато')
```

JavaScript. События