Javascript

Javascript- что это и для чего

-) JavaScript — язык сценариев, или скриптов. Скрипт представляет собой программный код — набор инструкций, который не требует предварительной обработки (например, компиляции) перед запуском. Код JavaScript интерпретируется движком браузера во время загрузки веб-страницы. Интерпретатор браузера выполняет построчный анализ, обработку и выполнение исходной программы или запроса.
- 2) JavaScript — объектно-ориентированный язык с прототипным наследованием. Он поддерживает несколько встроенных объектов, а также позволяет создавать или удалять свои собственные (пользовательские) объекты. Объекты могут наследовать свойства непосредственно друг от друга, образуя цепочку объект-прототип.

- для создания приложений для смартфонов и планшетов
- для разработки приложений для этих Windows
- для программирования самых различных "умных" устройств, которые взаимодействуют с интернетом

Javascript- что можно делать

- Динамически изменять содержимое веб-страниц;
- Привязывать к элементам обработчики событий (функции которые выполнят свой код только после того, как совершатся определенные действия);
- Выполнять код через заданные промежутки времени;
- Управлять поведением браузера (*открывать новые окна, загружать указанные документы и т. д.*);
- Создавать и считывать cookies;
- Определять, какой браузер использует пользователь (*также можно определить ОС, разрешение экрана, предыдущие страницы, которые посещал пользователь и т.д.*);
- Проверять данные форм перед отправкой их на сервер и многое другое.

Подключение Javascript

Сценарии JavaScript бывают **встроенные**, т.е. их содержимое является частью документа, и **внешние**, хранящиеся в отдельном файле с расширением .js. Сценарии можно внедрить в html-документ следующими способами:

- Для этого нужно разместить код в отдельном файле и включить ссылку на файл в заголовок
- *<head><script src="script.js"></script></head> или тело страницы.*
- *<body><script src="script.js"></script></body>*

Обработчик события

- **2)В виде обработчика события**.
- Каждый html-элемент имеет JavaScript-события, которые срабатывают в определенный момент. Нужно добавить необходимое событие в html-элемент как атрибут, а в качестве значения этого атрибута указать требуемую функцию. Функция, вызываемая в ответ на срабатывание события, является **обработчиком события**. В результате срабатывания события исполнится связанный с ним код. Этот способ применяется в основном для коротких сценариев, например, можно установить смену цвета фона при нажатии на кнопку:

```
<script>var colorArray = ["#5A9C6E", "#A8BF5A", "#FAC46E", "#FAD5BB", "#F2FEFF"]; // создаем массив с цветами фонаvar i = 0; function changeColor(){ document.body.style.background = colorArray[i];     i++;    if( i > colorArray.length - 1){       i = 0; }}</script><button onclick="changeColor();">Change background</button>
```

Внутрь тега <script>

- Элемент <script> может вставляться в любое место документа. Внутри тега располагается код, который выполняется сразу после прочтения браузером, или содержит описание функции, которая выполняется в момент ее вызова. Описание функции можно располагать в любом месте, главное, чтобы к моменту ее вызова код функции уже был загружен.
- Обычно код JavaScript размещается в заголовке документа (элемент <head>) или после открывающего тега <body>. Если скрипт используется после загрузки страницы, например, код счетчика, то его лучше разместить в конце документа:

<footer><script>document.write("Введите свое имя");</script></footer></body>

Выполнение кода javascript

- Когда браузер получает веб-страницу с кодом html и javascript, то он ее интерпретирует. Результат интерпретации в виде различных элементов - кнопок, полей ввода, текстовых блоков и т.д., мы видим перед собой в браузере. Интерпретация веб-страницы происходит последовательно сверху вниз.
- Когда браузер встречает на веб-странице элемент <script> с кодом javascript, то вступает в действие встроенный интерпретатор javascript. И пока он не закончит свою работу, дальше интерпретация веб-страницы не идет.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>JavaScript</title>
    <script>
    alert("Секция head");
    </script>
</head>
<body>
    <h2>Первый заголовок</h2>
    <script>
    alert("Первый заголовок");
    </script>
    <h2>Второй заголовок</h2>
    <script>
    alert("Второй заголовок");
    </script>
</body>
</html>
```

Основы синтаксиса JavaScript

Код javascript состоит из инструкций, каждая из которых завершается точкой запятой:

alert("Вычисление выражения"); var a = 5 + 8; alert(a);

Другое написание кода считается дурным тоном и снижает зарплату.

Однако современные браузеры вполне могут различать отдельные инструкции, если они просто располагаются на отдельных строках без точки запятой:

- *alert("Вычисление выражения")*
- *var a = 5 + 8*
- *alert(a)*

Комментарии

Комментарии могут быть однострочными, для которых используется двойной слеш:

- *// вывод сообщения*
- *alert("Вычисление выражения");*
- *// арифметическая операция*
- *var a = 5 + 8;*
- *alert(a);*

Кроме однострочных комментариев могут использоваться и многострочные. Такие комментарии заключаются между символами /*текст комментария*/. Например:

- */* вывод сообщения и*
- *арифметическая операция */*
- *alert("Вычисление выражения");*
- *var a = 5 + 8;*
- *alert(a);*

Чувствительность к регистру

- JavaScript – это язык, чувствительный к регистру символов. Это значит, что ключевые слова, имена переменных и функций и любые другие идентификаторы языка должны всегда содержать одинаковые наборы прописных и строчных букв.
- Например, ключевое слово while должно набираться как «while», а не «While » или «WHILE». Аналогично myvar, Myvar, MyVar и MYVAR – это имена четырех разных переменных.
- HTML и клиентский JavaScript тесно связаны, это различие может привести к путанице. Многие JavaScript-объекты и их свойства имеют те же имена, что и теги и атрибуты языка HTML, которые они обозначают. Однако если в HTML эти теги и атрибуты могут набираться в любом регистре, то в JavaScript они обычно должны набираться строчными буквами.
- Например, атрибут onclick обработчика события чаще всего задается в HTML как onClick, однако в JavaScript-коде (или в XHTML-документе) он должен быть обозначен как onclick.

Идентификаторы и зарезервированные слова

символы подчеркивания или знаки доллара. (Цифра не может быть первым символом Идентификатор - это просто имя. В JavaScript идентификаторы выступают в качестве имен переменных и функций, а также меток некоторых циклов. Идентификаторы в JavaScript должны начинаться с буквы, с символа подчеркивания (_) или знака доллара ($). Далее могут следовать любые буквы, цифры,, так как тогда интерпретатору трудно будет отличать идентификаторы от чисел.) Примеры допустимых идентификаторов:

- *I*
- *my_variable_name*
- *V13*
- *_myvar*
- *$str*

Метод document.write

метод document.write() пишет информацию на веб-страницу.

Заменим скрипт на такой

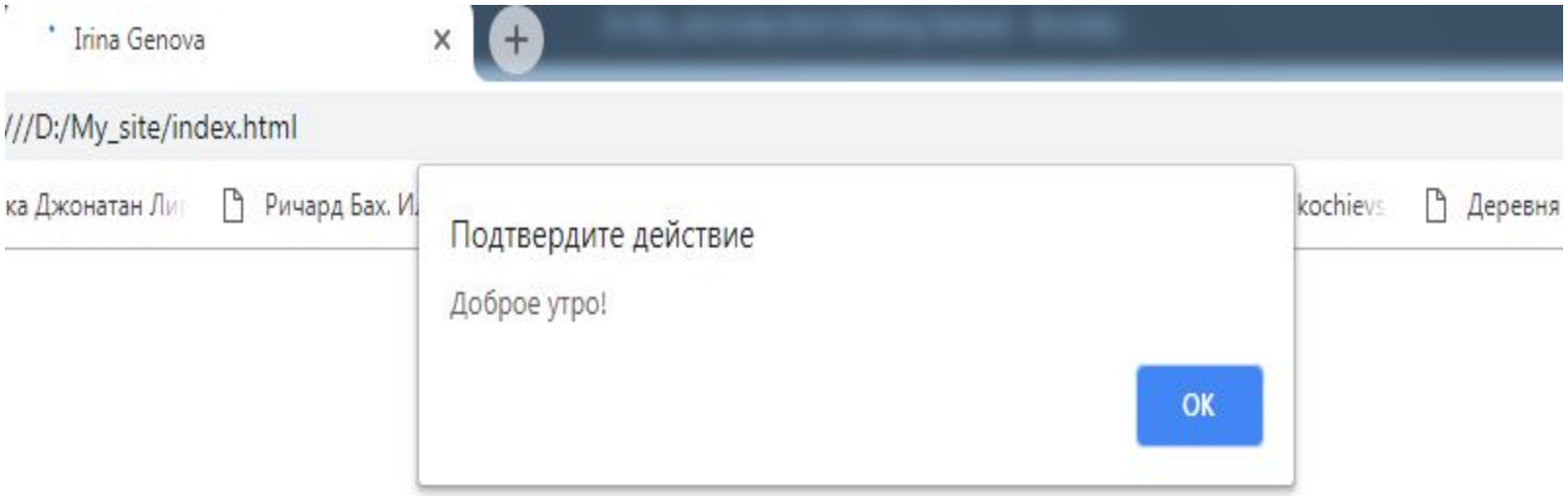
- *<script>*
- *var a = 5 + 8;*
- *document.write("Результат операции ");*
- *document.write(a);*
- *</script>*

Для совместимости и простоты редактирования для составления идентификаторов обычно используются только символы ASCII и цифры. Однако JavaScript допускает возможность использования в идентификаторах букв и цифр из полного набора символов Юникода. Это позволяет программистам давать переменным имена на своих родных языках и использовать в них математические символы.

JavaScript резервирует ряд идентификаторов, которые играют роль ключевых слов самого языка. Эти ключевые слова не могут служить идентификаторами в программах. JavaScript также резервирует некоторые ключевые слова, которые в настоящее время не являются частью языка, но которые могут войти в его состав в будущих версиях. В приведенной таблице перечислены все ключевые слова по категориям:

Зарезервированные ключевые слова JavaScript	
Категория	**Ключевые слова**
Базовые идентификаторы	break delete function return typeof case do if switch var catch else in this void continue false instanceof throw while debugger finally new true with default for null try
Новые ключевые слова в стандарте EcmaScript 5	class const enum export extends import super
Зарезервированные слова в строгом режиме (в обычном они доступны)	implements let private public yield interface package protected static arguments eval
Ключевые слова языка Java (зарезервированы в EcmaScript 3)	abstract double goto native static boolean enum implements package super byte export import private synchronized char extends int protected throws class final interface public transient const float long short volatile
Предопределенные глобальные переменные и функции	arguments encodeURI Infinity Number RegExp Array encodeURIComponent isFinite Object String Boolean Error isNaN parseFloat SyntaxError Date eval JSON parseInt TypeError decodeURI EvalError Math RangeError undefined decodeURIComponent Function NaN ReferenceError URIError

Как можно написать такую минипрограмму



Рассмотрим подробнее

Откроем файл *myscript.js* в текстовом редакторе и определим в нем следующий код:

```
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
```

Первое выражение получает текущую дату и присваивает ее переменной date. С помощью второй инструкции получаем время в часах. Дальше мы сравниваем полученное время с числом 13 и в зависимости от результатов проверки выводим первое или второе сообщение.

Зачем консоль

Разработчики используют консоль для отладки скриптов

Для вывода различного рода информации в консоли браузера используется специальная функция **console.log()**. Например, определим следующую веб-страницу:

```
<!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>
```

Переменная

Для хранения данных в программе используются переменные. Переменные предназначены для хранения каких-нибудь временных данных или таких данных, которые в процессе работы могут менять свое значение. Для создания переменных применяется ключевое слово var.

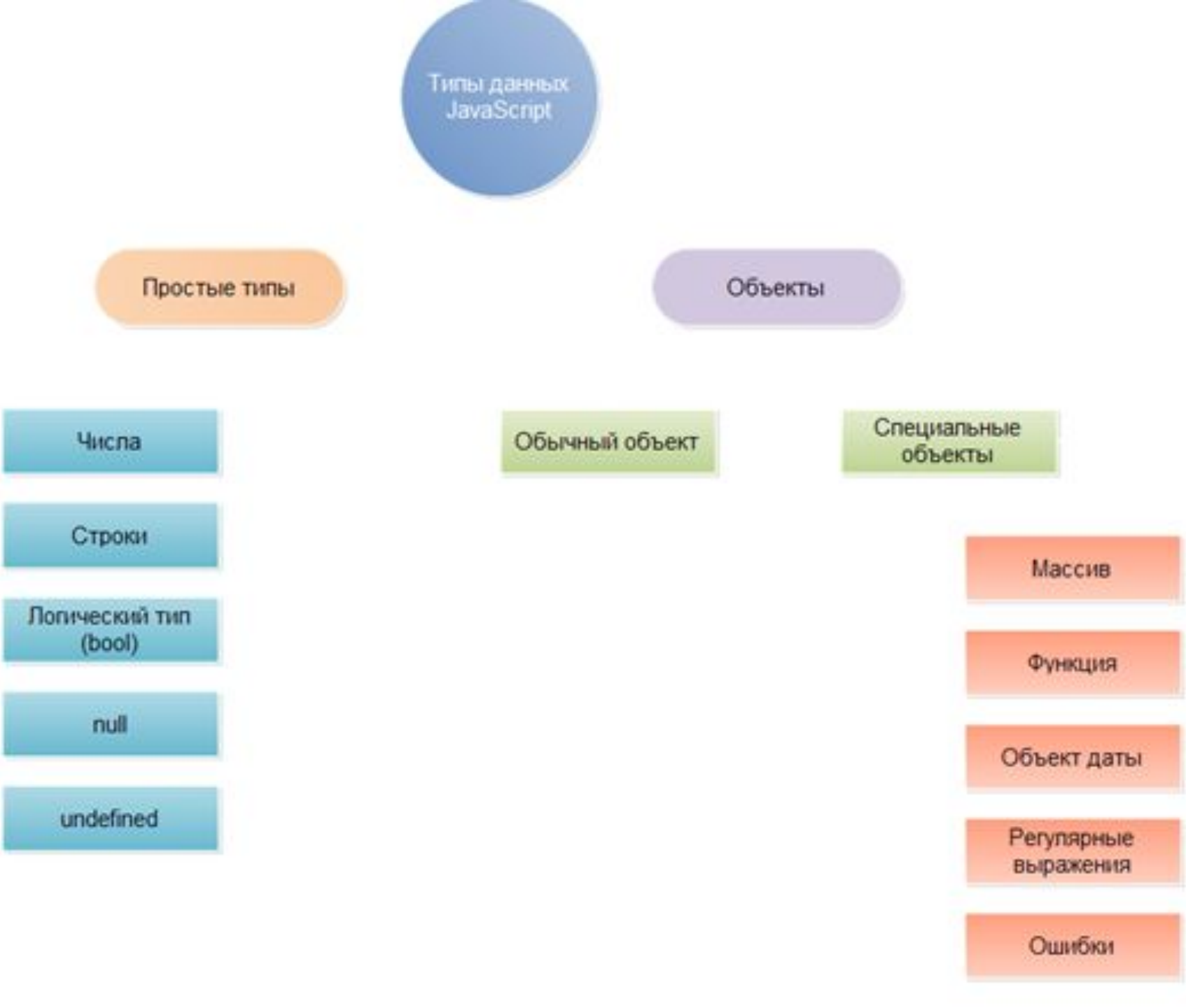
- Переменные по области видимости делятся на **глобальные** и **локальные**. **Область видимости** представляет собой часть сценария, в пределах которой имя переменной связано с этой переменной и возвращает ее значение. Переменные, объявленные внутри тела функции, называются **локальными**, их можно использовать только в этой функции. Локальные переменные создаются и уничтожаются вместе с соответствующей функцией.
- Переменные, объявленные внутри элемента <script>, или внутри функции, но без использования ключевого слова var, называются **глобальными**. Доступ к ним может осуществляться на протяжении всего времени, пока страница загружена в браузере. Такие переменные могут использоваться всеми функциями, позволяя им обмениваться данными.

Важно

- Если глобальная и локальная переменная имеют одинаковые имена, то локальная переменная будет иметь преимущество перед глобальной.
- Локальные переменные, объявленные внутри функции в разных блоках кода, имеют одинаковые области видимости. Тем не менее, рекомендуется помещать объявления всех переменных в начале функции.

Внутри тела функции локальная переменная имеет преимущество перед глобальной переменной с тем же именем. Если объявить локальную переменную или параметр функции с тем же именем, что у глобальной переменной, то фактически глобальная переменная будет скрыта:

```
var result = 'global';
function getResult() {
  var result = 'local';
  return result;  };
console.log(getResult());    // Отобразит 'local'
```


Типы данных JavaScript
Простые типы
Объекты
Числа
Строки
Логический тип (bool)
null
undefined
Обычный объект
Специальные объекты
Массив
Функция
Объект даты
Регулярные выражения
Ошибки

Типы данных

Все используемые данные в javascript имеют определенный тип. 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

- Специальные значения null и undefined являются элементарными значениями, но они не относятся ни к числам, ни к строкам, ни к логическим значениям. Каждое из них определяет только одно значение своего собственного специального типа.

Объекты

Любое значение в языке JavaScript, не являющееся числом, строкой, логическим значением или специальным значением null или undefined, является объектом.

- ***Объект*** (т.е. член объектного типа данных) представляет собой коллекцию свойств, каждое из которых имеет имя и значение (либо простого типа, такое как число или строка, либо объектного).
- Обычный объект JavaScript представляет собой неупорядоченную коллекцию именованных значений. Кроме того, в JavaScript имеется объект специального типа, известный как **массив**, представляющий упорядоченную коллекцию пронумерованных значений. Для работы с массивами в языке JavaScript имеются специальные синтаксические конструкции.

Специальный тип объекта

Это функция.

Функция - это объект, с которым связан выполняемый код. Функция может вызываться для выполнения определенной операции и возвращать вычисленное значение. Подобно массивам, функции ведут себя не так, как другие виды объектов, и в JavaScript определен специальный синтаксис для работы с ними. Одна из важнейших особенностей функций в JavaScript состоит в том, что они являются самыми настоящими значениями, и программы JavaScript могут манипулировать ими, как обычными объектами.

Функции, которые пишутся для инициализации вновь создаваемых объектов (с оператором new), называются *конструкторами*.

Каждый конструктор определяет ***класс объектов*** - множество объектов, инициализируемых этим конструктором. Классы можно представлять как подтипы объектного типа.

В дополнение к классам **Array** и **Function** в базовом языке JavaScript определены еще три полезных класса. Класс **Date** определяет объекты, представляющие даты. Класс **RegExp** определяет объекты, представляющие регулярные выражения (мощный инструмент сопоставления с шаблоном). А класс Error определяет объекты, представляющие синтаксические ошибки и ошибки времени выполнения, которые могут возникать в программах на языке JavaScript. Имеется возможность определять собственные классы объектов, объявляя соответствующие функции-конструкторы.

Числа

Все числа в JavaScript представляются вещественными значениями (с плавающей точкой). Для представления чисел в JavaScript используется 64-битный формат, определяемый стандартом *IEEE 754*. Этот формат способен представлять числа в диапазоне от ±1,8 x 10^{308} до ±5 x 10^{-324}.

В JavaScript целые десятичные числа записываются как последовательность цифр. Помимо десятичных целых литералов JavaScript распознает шестнадцатеричные значения. Шестнадцатеричные литералы начинаются с последовательности символов «0x», за которой следует строка шестнадцатеричных цифр.

. Шестнадцатеричная цифра - это одна из цифр от 0 до 9 или букв от A до F, представляющих значения от 10 до 15:

var a = 255;

var b = 0xFF; // Число 255 в шестнадцатеричной системе исчисления

- Литералы вещественных чисел должны иметь десятичную точку - при определении таких литералов используется традиционный синтаксис вещественных чисел. Вещественное значение представляется как целая часть числа, за которой следуют десятичная точка и дробная часть числа.
- **(M24)** Литералы вещественных чисел могут также представляться в экспоненциальной нотации: вещественное число, за которым следует буква e (или E), а затем необязательный знак плюс или минус и целая экспонента. Такая форма записи обозначает вещественное число, умноженное на 10 в степени, определяемой значением экспоненты:
- *var a = 16.75;*
- *var b = 2e4; // 2 * 10^4 = 20 000*

Арифметические операции

Обработка чисел в языке JavaScript выполняется с помощью арифметических операторов. В число таких операторов входят: оператор сложения +, оператор вычитания - , оператор умножения *, оператор деления / и оператор деления по модулю % (возвращает остаток от деления).

Помимо этих простых арифметических операторов JavaScript поддерживает более сложные математические операции, с помощью функций и констант, доступных в виде свойств объекта **Math**:

```
Math.pow(2,53)          // 2 в степени 53
Math.round(.6)          // Округление до ближайшего целого (результат 1.0)
Math.ceil(.6)           // Округление вверх (результат 1.0)
Math.floor(.6)          // Округление вниз (результат 0)
Math.abs(-5)            // Модуль числа (результат 5)

Math.sqrt(3)            // Корень квадратный из 3
```

Потеря разрядов

Арифметические операции в JavaScript не возбуждают ошибку в случае переполнения, потери значащих разрядов или деления на ноль. Если результат арифметической операции окажется больше самого большого представимого значения (переполнение), возвращается специальное значение «бесконечность», которое в JavaScript обозначается как **Infinity**. Аналогично, если абсолютное значение отрицательного результата окажется больше самого большого представимого значения, возвращается значение «отрицательная бесконечность», которое обозначается как *-Infinity*.

Деление на ноль

Деление на ноль не считается ошибкой в JavaScript: в этом случае просто возвращается бесконечность или отрицательная бесконечность. Однако есть одно исключение: операция деления нуля на ноль не имеет четко определенного значения, поэтому в качестве результата такой операции возвращается специальное значение «не число» (not-a-number), которое обозначается как **NaN**. Значение NaN возвращается также при попытке разделить бесконечность на бесконечность, извлечь квадратный корень из отрицательного числа или выполнить арифметическую операцию с нечисловыми операндами, которые не могут быть преобразованы в числа.

Операции с переменными

JavaScript поддерживает все базовые математические операции:

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;
```

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
```

Результатом будет 5, так как наибольшее целое число, которое меньше или равно 40 и при этом делится на 7 равно 35, а 40 - 35 = 5.

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:

```
// 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
```

Постфиксный инкремент аналогичен операции:

```
a = a + 1; // a++
```

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
```

Как и принято в математике, 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
```