Оглавление

Переменные JavaScript

Создание переменных в JavaScript

Переменная в JavaScript — это *именованная* область в оперативной памяти компьютера, которая хранит в себе какие-либо данные: числа, строки (текст) и т.д. У каждой переменной имеется своё уникальное имя *(которое вы придумываете сами)*, обращаясь к переменной по имени, можно получить данные которая она в себе хранит.

Создание переменной

Переменные в JavaScript создаются следующим образом:

var имя Переменной;

Сначала пишут **ключевое слово** `var`, затем пробел и *имя переменой*, в конце строки ставят точку с запятой;

Имена переменных в JavaScript

Имена **переменным в JavaScript**, вы придумываете самостоятельно.

Правила задания имён переменных:

- имя переменной может состоять из любых цифр и букв английского алфавита,
- в имени переменной, могут использоваться символы доллара `$` и подчёркивания
- имя переменной, не может начинаться с цифры,
- в качестве имён переменных нельзя использовать ключевые слова и зарезервированные слова.

Приведём примеры правильных и не правильных имён переменных:

Правильно	Не правильно
privet	pri vet
$priVet	5priVet
_privet	pri-vet
pri_vet	pri?vet
PRIVET	+privet+

Присвоение переменной значения

Чтобы **переменная**, начала хранить внутри себя данные, ей нужно присвоить какое-либо **значение**:

Имя Переменной = значение переменной;

Сначала записывают имя переменной, затем `=` (знак присваивания) и `значение переменной`, в конце строки ставят точку с запятой;

Давайте создадим переменную и присвоим ей значение `Россия` (если вы из другой страны, то можете написать имя своего государства).

```
var myCountry;
myCountry = "Россия";
```

В первой строке, создали переменную с именем `myCountry`, во второй строке, присвоили переменной `myCountry`, значение `Россия`.

Обратие внимание на то, что значение `Россия` нужно обрамить кавычками. Кавычки используют тогда, когда значение имеет **строковый тип данных**. Кавычки не используют, когда значение имеет **числовой тип данных**. О существующих типах данных в языке JavaScript, мы поговорим с вами на следующем уроке.

Получаем доступ к значению переменной

Чтобы получить доступ к **значению** (данным) переменной, нужно обратиться к переменной по имени:

```
var myCountry; // создали переменную
myCountry = "Россия"; // присвоили переменной значение
alert (myCountry); // обратились к переменной
```

Добавили к нашей программе (скрипту) третью строку, в которой в функцию `alert()`поместили имя переменной, **без кавычек**, в результате окошко покажет нам надпись `Россия`.

Объявление переменной, инициализация переменной

Обычно программисты вместо словосочетаний создание переменной и присвоение переменной значения, используют словосочетания объявление переменной и инициализация переменной.

Объявление переменной — это создание переменной.

Инициализация переменной — это присвоение переменной значения.

Типы данных JavaScript

Какие типы данных существуют в JavaScript?

В прошлом уроке мы выяснили, что переменная — это именованная область памяти, которая хранит в себе какие-либо данные (значения).

У каждого значения в JavaScript, имеется свой **тип данных**. Всего в JavaScript существует 6 типов данных, в этом уроке по JavaScript, мы рассмотрим 4 типа данных:

- числовой тип данных `number`,
- строковый тип данных `string`,
- логический тип данных `boolean`,
- неопределённый тип данных `undefined`.

Остальные два изучим чуть позже:

- объектный тип данных `object`
- пустой тип данных `null`

Оператор typeof

Прежде чем рассмотреть **типы данных JavaScript**, познакомимся сначала с оператором `typeof`, он позволяет узнать какой тип данных присвоен переменной, делается это следующим образом:

alert(typeof имяПеременной);

После этого скрипт должен выдать какое-либо сообщение: `number`, `string`, `boolean`, `undefined`, `object`.

Тип данных: число (number)

Когда переменной, в качестве значения, присваивается какое-либо число *(без кавычек)*, её тип данных становится `number`

```
var myNumber;
myNumber = 5;
alert(typeof myNumber);
```

В первой строке мы создали переменную с именем `myNumber`, во второй строке, присвоили переменной значение `5`, в третьей с помощью оператора `typeof` вычислили тип данных у переменной `myNumber`, а функция `alert()` показала нам результат этих вычислений.

В итоге, данный скрипт выведет нам сообщение `number`. Если число обрамить кавычками *(одинарными '5' или двойными "5")*, то оно превратится в строку `string`.

Тип данных: строка (string)

Когда переменной в качестве значения присваивается какое-либо значение, заключённое в двойные `" "` или одинарные кавычки `' '`, то её тип данных становится `string`.

```
var myString;
myString = "Привет, я строка JavaScript!";
alert(typeof myString);
```

В первой строке мы создали переменную с именем `myString`, во второй строке, присвоили переменной значение `"Привет, я строка JavaScript!"`, в третьей с помощью оператора `typeof` вычислили тип данных у переменной `myString`, а функция `alert()` показала нам результат этих вычислений. В итоге, данный скрипт должен вывести нам сообщение `string`.

Тип данных: логический тип данных (boolean)

Когда переменной в качестве значения присваивается `true` или `false`, **без кавычек**, то её тип данных становится `boolean`.

Тип данных `boolean` — это логический тип данных, он имеет всего два значения: `true`*(правда)* или `false` *(ложь)*.

```
var myBoolean;
myBoolean = true;
alert(typeof myBoolean);
```

В первой строке мы создали переменную с именем `myBoolean`, во второй строке, присвоили переменной значение `true`, в третьей с помощью оператора `typeof`вычислили тип данных у переменной `myBoolean`, а функция `alert()` показала нам результат этих вычислений. В итоге, данный скрипт должен вывести нам сообщение `boolean`.

Логический тип данных, мы изучим более подробно в следующих уроках, посвящённых операциям сравнения, логическим операциям и оператору ветвления `if`

Тип данных: undefined

Тип данных `undefined` появляется тогда, когда переменная объявлена, но не инициализирована, т.е. переменная создана, но значения ей не присвоили.

```
var myUndefined;
alert(typeof myUndefined);
```

В первой строке мы создали переменную с именем `myUndefined`, во второй строке с помощью оператора `typeof` вычислили тип данных у переменной `myUndefined`, а функция `alert()` показала нам результат этих вычислений. В итоге, данный скрипт должен вывести нам сообщение `undefined`.

Доступ к значению переменной

Чтобы получить доступ, к значению переменной, нужно обратиться к ней по имени:

```
// объявляем переменные
var myString;
var myNumber;


// инициализируем переменные
```

```
myString = "Привет, МИР!";
myNumber = 5;


// обращаемся к переменным
alert(myString);
alert(myNumber);
```

В первой и второй строках кода мы создали переменные `myString` и `myNumber`, в третьей и четвёртой строках присвоили переменным значения `"Привет, МИР!"` и `5`, в пятой и шестой строках с помощью функции `alert()` вывели результаты `Привет, МИР!` и `5`

На данном этапе, ознакомление с типами данных в JavaScript можно закончить, далее приступим к изучению операций в JavaScript.

Операции JavaScript

Какие операции существуют в JavaScript?

Операций в JavaScript, существует 6 видов, в этом уроке мы рассмотрим 4 основных вида операций:

1. Операция присваивания *(мы уже с ней знакомы)*,
2. Арифметические операции,
3. Операции сравнения,
4. Логические операции.

Операция присваивания JavaScript

Когда мы инициализируем переменную (присваиваем ей значение) в JavaScript, мы используем **операцию присваивания** `=`. В языке JavaScript знак `=` означает *присваивание*.

```
var helloWorld;
helloWorld = "Привет Мир!";
```

Во второй строке, переменной `helloWorld`, с помощью операции присваивания `=`, было присвоено значение `"Привет Мир!"`

Арифметические операции в JavaScript

Арифметические операции — это обычные математические операции, которые все мы изучали в школе:

`+` операция сложения,

`-` операция вычитания,

`*` операция умножения,

`/` операция деления.

Напишем небольшую программу для сложения двух чисел:

```
var numberOne;
var numberTwo;
var numberSum;
numberOne = 5;
numberTwo = 7;
numberSum = numberOne + numberTwo;
```

```
alert(numberSum);
```

Создали три переменные `numberOne`, `numberTwo` и `numberSum`. Первым двум переменным `numberOne` и `numberTwo` присвоили значения `5` и `7`, третьей переменной `numberSum` присвоили **операцию сложения** `+` с двумя первыми переменными. Эта операция `numberOne + numberTwo` (5 + 7), присвоит переменной `numberSum`значение `12`

В последней строке, получили доступ к значению переменной `numberSum`, передав её в функцию `alert()`. В итоге у вас должно появится окошко с сообщением `12`

Поэкспериментируйте немного с этим скриптом, подставляя вместо операции сложения `+`другие арифметические операции, а вместо значений `5` и `7`, другие числа.

Операции сравнения в JavaScipt

Операции сравнения применяются тогда, когда нужно сравнить между собой, значения двух переменных или просто два значения. Операция сравнения в зависимости от верности (правда) или неверности (ложь) сравнения, возвращает булевый тип данных: `true` или `false`.

Рассмотрим операции сравнения JavaScript:

`==` операция равенства,

`===` операция строго равенства,

`>` операция больше,

`<` операция меньше,

`>=` операция больше или равно,

`<=` операция меньше или равно,

`!=` операция неравенства,

`!==` операция строго неравенства.

Рассмотрим примеры сравнения чисел:

`alert(5 == 5);` — сообщение покажет `true`,

`alert(5 == 7);` — сообщение покажет `false`,

`alert(5 > 7);` — сообщение покажет `false`,

`alert(5 < 7);` — сообщение покажет `true`,

`alert(5 >= 7);` — сообщение покажет `false`,

`alert(5 <= 7);` — сообщение покажет `true`,

`alert(5 >= 5);` — сообщение покажет `true`,

`alert(5 != 5);` — сообщение покажет `false`,

`alert(5 != 7);` — сообщение покажет `true`.

Операции сравнения обычно применяются в операторах ветвления if, где в зависимсоти от того, что вернет операция `true` или `false`, запустится тот или иной блок кода.

Логические операции

Логические операции, анализируют результаты операций сравнения и на основе этих данных, выдают `true` или `false`.

Схема логической операции:

(операция равнения) логическая операция (операция сравения)

Рассмотрим логические операции JavaScript:

1. `&&` логическая операция **И**, если обе операции сравнения вернут `true`, то логическая операция **И** тоже вернёт `true`, если хотя бы одна из операций сравнения вернёт `false`, то тогда и логическая операция **И** тоже вернёт `false`
2. `||` логическая операция **ИЛИ**, если одна или обе операции сравнения вернут `true`, то логическая операция **ИЛИ** тоже вернёт `true`, если обе операции сравнения вернут `false`, то тогда и логическая операция **ИЛИ** тоже вернёт `false`

Примеры логических операций:

```
(2 > 1)&&(4 == 4)
```

Логическая операция `&&`, вернет `true`, поскольку обе операции сравнения верны и возвращают `true`.

```
(2 < 1)&&(4 == 4)
```

Логическая операция `&&`, вернет `false`, поскольку одна из операций сравнения `2 < 1`не верна и возвращает `false`.

```
(2 < 1)&&(4 == 3)
```

Логическая операция `&&`, вернет `false`, поскольку обе операции сравнения не верны и возвращают `false`.

```
(2 > 1)||(4 == 4)
```

Логическая операция `||`, вернет `true`, поскольку обе операции сравнения верны и возвращают `true`.

```
(2 > 1)||(4 == 3)
```

Логическая операция `||`, вернет `true`, поскольку одна из операций сравнения `2 > 1`верна и возвращает `true`.

```
(2 < 1)||(4 == 3)
```

Логическая операция `||`, вернет `false`, поскольку обе операции сравнения не верны и возвращают `false`.

Операторы if JavaScript

Операторы ветвления предназначены для того, чтобы программа могла запускать тот или иной блок кода, в зависимости от верности `true` или не верности `false` условия.

Операторов ветвления существует пять видов, в этом уроке мы рассмотрим два из них:

- оператор ветвления `if`
- оператор ветвления `if else`

Оператор ветвления if

Оператор ветвления if запускает код, если условие возвращает `true`.

В качестве условия, в операторах ветвления, обычно выступают операции сравнения или логические операции.

Схема оператора ветвления `if`, выглядит следующим образом:

```
if (условие) {
 код запустится, если условие вернёт true
}
```

Приведём пример с оператором ветвления `if`:

```
// создадим две переменные
var numOne;
var numTwo;
```

```
// присвоим переменным значения
numOne = 5;
numTwo = 3;

if (numOne > numTwo) {
 alert("Условие возвратило true");
}
```

В скрипте мы создали две переменные `numOne` и `numTwo`, присвоили им числовые значения `5` и `3`.

Далее создали оператор ветвления `if`, который сравнивает между собой значения двух переменных. Если операция сравнения вернёт `true`, то запустится код расположенный между фигурными скобками. В нашем случае, появится окошко с сообщением `Условие возвратило true`. Если операция сравнения вернёт `false`, то ничего не произойдёт.

Символы двойного слеша `//`, являются комментарием. После двойного слэша можно написать любой текст, интерпретатор языка JavaScript, будет воспринимать его как комментарий и обрабатывать не будет. Как мы помним, в языках HTML и CSS, тоже можно создавать комментарии.

Прочитать подробнее о комментариях в JavaScript.

Оператор ветвления if else

Оператор ветвления if else, предназначен для запуска того или иного блока кода, в зависимости от значения которое вернёт условие: `true` или `false`

Схема оператора ветвления `if else`, выглядит следующим образом:

```
if (условие) {
 код запустится, если условие вернёт true
} else {
 код запустится, если условие вернёт false
}
```

Приведём пример с оператором ветвления `if else`:

```
var numOne;
var numTwo;

numOne = 5;
numTwo = 3;

if (numOne > numTwo) {
 alert("Условие возвратило true");
} else {
 alert("Условие возвратило false");
}
```

Присвойте переменной `numTwo`, число большее чем `5`, например `7`, тогда условие вернёт `false` и появится окошко с сообщением `Условие возвратило false`.

Оператор ветвления также имеет и другие названия: условный оператор, условная конструкция, условная инструкция.

Цикл for JavaScript

Что такое цикл for в JavaScript

Циклы предназначены для того, чтобы выполнять один и тот же блок кода, множество раз.

Всего в JavaScript существует четыре вида циклов, самый распространённый из них **цикл for**, его мы и будем рассматривать в данном уроке, а с остальными вы можете ознакомиться в дополнительных статьях посвященным циклам JavaScript.

Схема цикла for:

```
var i;
for(i=0; условие; i++) {
 блок кода;
}
```

В качестве условия, в **циклах JavaScript**, обычно выступают операции сравнения.

Пример использования цикла:

```
var i;
for(i=0; i<5; i++) {
 alert("Привет!");
}
```

Данный скрипт, выведет пять окошек подряд, с надписью `Привет!`. Рассмотрим каждую часть цикла.

Сначала создаём переменную `var i;`, которая будет участвовать в цикле, меняя свое значение. Затем на следующей строке пишем `for() { }`, где между фигурными скобками `{ }` располагаем код, который будет исполняться определённое количество раз.

Между обычными скобками `()` располагаем:

`i=0;` присвоили начальное значение, переменной `i`,

`i<5;` условие, при котором будет работать цикл,

`i++` арифметическая операция, увеличивающая переменную `i` на единицу `1`, при каждом проходе цикла.

В итоге получаем, что всего в цикле будет 5 проходов (итераций), окошко появится 5 раз.

Разбираем подробней работу цикла for

Интерпретатор JavaScript, находя в коде цикл `for`, действует следующим образом, сначала он выясняет значение переменной `i` в нашем примере `i=0`, далее проверяется условие `i<5`, если условие вернёт `true`, то запустится код размещённый в фигурных скобках `{ }`. После чего выполнится арифметическая операция `i++`, т.е. `i + 1`.

Затем все повторяется заново:

определение значения переменной `i`,

проверка условия,

выполнение кода,

арифметическая операция `i++`

Один проход по циклу, называется *итерацией*.

В нашем примере, будет всего 5 итераций, со значениями `i` равными `0`, `1`, `2`, `3`, `4`. Как только `i` станет равной `5`, то условие `i<5` вернёт `false` и выполнение цикла прекратится.

Функции JavaScript

Что такое пользовательская функция в JavaScript?

Функции — это блоки кода, которые имеют своё имя и их можно вызывать по этому имени.

Функции в JavaScript бывают как встроенными например `alert()`, так и пользовательскими которые программист создаёт сам.

Пользовательская функция в JavaScript, создаётся следующим образом, сначала пишут ключевое слово `function` затем имя функции со скобками, после этого через пробел, ставят фигурные скобки и в них записывают необходимый код.

Схема пользовательской функции:

```
// создание функции
function имя_функции() {
 блок кода;
}


// вызов функции
имя_фнкции();
```

Основная цель функции заключается в том, чтобы избавить программу от дублирования кода. Записал один раз код в функцию и всё, можешь потом вызывать её по имени сколько угодно раз.

Имя функции не должно совпадать с именем какой-либо переменной. Правила задания имён функциям, такие же как и правила задания имён переменным.

Создадим пользовательскую функцию, которая будет выводить три окошка подряд.

```
function hello() {
 alert("Привет,");
 alert("как");
 alert("дела?");
}
```

Для того, чтобы выполнился код находящийся в функции, его нужно вызвать, для этого нужно написать имя функции со скобками, в конце не забудьте поставить точку с запятой.

```
hello();
```

Массивы JavaScript

Что такое массив в JavaScript?

Массив JavaScript — это набор различных значений, доступ к которым осуществляется по имени массива и индексу значения массива. Значения в массиве могут быть строками, числами, булевым типом данных и т.д.

Массив в JavaScript, создаётся с помощью литерала массива `[ ]`

Схема создания массива:

```
var имяМассива;
имяМассива = [];
```

Сначала создаём переменную, затем присваиваем переменной литерал массива, так создаётся пустой массив. Для того, чтобы заполнить массив, данными, нужно между квадратными скобками, через запятую разместить значения.

```
var имяМассива = [1, 2, 3];
```

Создали массив состоящий из трёх значений, которые имеют числовой тип данных.

```
var имяМассива = [1, "строка", true];
```

Создали массив, состоящий из трех значений, которые имеют числовой, строчный и логический (булев) тип данных.

Значения в массиве разделяются запятыми.

Получаем доступ к значениям массива

Создадим массив `capital`, с пятью значениями, в качестве которых будут выступать названия столиц различных государств:

```
var capital;
capital = ["Москва", "Берлин", "Пекин", "Лондон", "Токио"];
```

У каждого значения в массиве, имеется свой **числовой индекс**, благодаря чему к ним можно получить доступ. Числовые индексы начинаются с нуля.

У массива `capital` имеется 5 значений, первое значение — это `Москва`, его числовой индекс равен нулю `0`, второе значение — это `Берлин`, его числовой индекс равен единице `1` и т.д.

Чтобы получить доступ к значению массива, нужно написать имя массива с квадратными скобками, в которых должен находится числовой индекс значения:

```
capital[0]; // получили доступ к первому значению массива,
capital[1]; // получили доступ ко второму значению массива.
```

Пример работы с массивом:

```
var capital;
capital = ["Москва", "Берлин", "Пекин", "Лондон", "Токио"];
alert(capital[2]);
```

Создали переменную `capital`, присвоили ей массив с данными. Получили доступ к третьему значению массива и вывели его в виде сообщения, с помощью функции `alert()`. В итоге, у вас должно появится окошко с надписью `Пекин`.

Объекты JavaScript

Что такое пользовательские объекты в JavaScript?

Объект в JavaScript — это составной тип данных, который может хранить в себе числа, строки, булев тип данных и т.д. Также объекты могут хранить в себе функции и другие объекты.

Объект в JavaScript создаётся с помощью литерала объекта `{ }`

Схема создания объекта:

```
var имяОбъекта;
имяОбъекта = { };
```

Сначала создаём переменную, затем присваиваем переменной литерал объекта, так создаётся пустой объект. Для того, чтобы заполнить объект данными, нужно между фигурными скобками, через запятую, разместить элементы объекта.

```
var имяОбъекта;
имяОбъекта = {
 ключ: значение,
 ключ: значение,
 ключ: значение
};
```

Между фигурными скобками помещают **элементы объекта**, каждый элемент объекта делится на две части *ключ* и *значение*, разделённое двоеточием: Сами **элементы объекта** разделены друг от друга запятыми. После закрывающей фигурной скобки, нужно поставить точку с запятой.

`ключ: значение` — это элемент объекта.

`ключ` — это имя переменной,

`значение` — это значение переменной.

В качестве значений обычно выступают строки, числа, булев (логический) тип данных и т.д., также в качестве значений могут быть функции или другие объекты.

Давайте создадим, например, объект `cartman` и добавим в него элементы (Картман это персонаж из культового мультсериала «Южный Парк», а элементы в данном случае, будут характеристиками данного персонажа):

```
var cartman;
cartman = {
 gender: "мальчик",
 age: 10,
 physique: "толстый",
 character: "капризный",
 married: false,
 respect: function() {
  alert("Уважай, мою власть!!!");
 },
};
```

Создали объект `cartman`. Объект состоит из 6 элементов.

Имена ключей вы придумываете самостоятельно, как и их значения.

Методы и свойства объекта

Если в качестве значения ключа объекта, выступает строка, число, логический тип данных, объект и т.д. то его называют **свойством объекта**. Если в качестве значения ключа объекта выступает функция, то его называют **методом объекта**.

В нашем примере в объекте `cartman` присутствует 5 свойств и 1 метод `respect`

Получаем доступ к элементам объекта

Чтобы получить доступ к значению какого-либо ключа объекта, нужно написать имя объекта и через точку имя ключа.

```
имяОбъекта.имяКлюча
```

Получаем доступ к возрасту Картмана:

```
cartman.age;
```

Выводим возраст Картмана в окошке, с помощью функции `alert()`

```
alert(cartman.age);
```

Получаем доступ к методу `respect`, объекта `cartman`:

```
cartman.respect();
```

Так как метод это функция, то в конце имени ключа нужно ставить скобки `()`

В итоге, наш скрипт должен выглядеть следующим образом:

```
var cartman;
cartman = {
gender: "мальчик",
age: 10,
physique: "толстый",
character: "капризный",
married: false,
respect: function() {
alert("Уважай, мою власть!!!");
},
};
alert(cartman.age);
cartman.respect();
```

Лексическая структура JavaScript

Лексика — это словарный состав какого-либо языка.

Когда вы будете читать книги по JavaScript, вам часто будут попадаться незнакомые слова, являющиеся частью лексики данного языка. В этой статье, мы рассмотрим **лексическую структуру языка JavaScript**. Познакомимся с основными терминами и дадим им определения.

Термины и определения лексической структуры языка JavaScript:

Лексема *(токен)* — это минимальная часть языка, имеющая самостоятельный смысл. В JavaScript к лексемам относятся: разделители, идентификаторы, литералы, операции, операторы и т.д. Лексемы располагаются между разделителями (пробелами, табуляцией или переносом строки). Интерпретатор разделяет код программы на лексемы и затем собирает из них осмысленные конструкции. Примеры лексем:

```
var myLexeme; // тут 4 лексемы (var, пробел, myLexeme, точка с запятой)
```

```
myLexeme = "строка"; // тут 6 лексем
```

Идентификатор — это уникальное имя переменной, константы, пользовательской функции, объекта, массива, ключевых и зарезервированных слов, меток и т.д. Примеры идентификаторов:

```
var myIdentifier; // тут два идентификатора (var и myIdentifier)
function myIdentifier() { } // тут два идентификатора
```

Литерал — это значение переменной **заданное программистом**, оно может быть числом, строкой, логическим значением (true/false), регулярным выражением (для поиска по шаблону) и т.д. Значение созданное программой, литералом не является. Примеры литералов:

```
var myLiteral;
myLiteral = "строка";
myLiteral = число;
myLiteral = true;
```

Операнд — это левая или правая часть операции. Примеры операндов:

```
var myOperand;
myOperand = 8 + 2; // здесь в качестве операндов выступают myOperand, 8 и 2
```

Здесь в качестве операций выступают: `+` и `=`, в качестве выражений: `8 + 2` и `myOperand = 10`, в качестве строки кода (инструкции): `myOperand = 8 + 2;`

Операция — это специальные знаки которые манипулируют операндами, например: плюс `+`, минус `-`, больше `<`, меньше `>` и т.д. Примеры операций:

```
var myOperator;
myOperator = 7 + 3; // здесь в качестве операций выступают = и +
```

Операции могут быть арифметическими, сравнения, логическими и т.д.

Выражение — это комбинация операндов и операций, которая может быть вычислена интерпретатором для получения значения. Примеры выражений:

```
5 + 7 // + операция, 5 и 7 операнды
(6 - 2) * 5 // - и * операции, (6 - 2) и 5 операнды
```

Строка кода (команда, инструкция), указывает совершить какое либо действие, оканчивается точкой с запятой. Пример инструкции `x = y + z;` Разница между инструкцией и выражением, состоит в том, что выражение вычисляет, но ничего не делает, не изменяет программу, а строка кода изменяет.

Оператор — под операторами обычно имеют ввиду операторы ветвления: `if`, `else`, `switch`, операторы циклов: `for`, `while`, `do while`. Оператор ветвления также еще назвают: условной инструкцией, условным оператором, условной конструкцией.

Ключевое слово — это часть синтаксиса ядра языка, так называемый **предопределённый идентификатор**. Его нельзя использовать в качестве имени идентификатора. Примеры ключевых слов: `break, delete, function, if` и т.д.

Зарезервированное слово — это часть синтаксиса ядра языка, которое **планируется использовать в будущем**, использовать их в качестве идентификатора не рекомендуется. Примеры зарезервированных слов: `const, import, int, long` и т.д.

Дополнительный материал

Объявление переменной — это создание переменной.

Инициализация переменной — это присваивание переменной какого-либо значения.

Тип данных — это значение присваиваемой переменной, может быть числом, строкой, булевым выражением (true, false) и т.д.

NaN — Not a Number (не число), значение (тип данных) не являющееся числом. NaN появляется, когда в вычислениях появляется ошибка связанная с тем, что один из операндов арифметической операции не имеет числового типа данных. Пример:

```
var myVariable = "Строка";
alert(5 * myVariable); // выведет NaN
```

undefined (неопределённая) — состояние не инициализированной переменной. Появляется когда в сценарии участвует переменная которой не присвоили значения. Пример:

```
var myVariable;
alert(myVariable); // выведет undefined
```

Переменная объявлена, но не инициализирована, значение у неё `undefined`. В логическом контексте `undefined` принимает значение `false`, в числовом `NaN`, в строковом `"undefined"`

null — это значение которое не имеет типа данных. Например null может появится, при нажатии в модальном окне на кнопку *«Отмена»*, данное окно вызывается с помощью функции `promt();`. В логическом контексте, nul принимает значение `false`, в числовом `0`, в строковом `"null"`

Конкатенация — это операция соединения строк, в качестве операции выступает знак плюс `+`. Для того чтобы произошла операция конкатенации, нужно чтобы один или два операнда имели тип данных `string`. Пример:

```
alert("5" + 3);
```

Данная функция выведет «53», а не 8, поскольку один из операндов имеет тип данных, строка.

Интерпретатор — это специальная программа, которая понимает код языка и выполняет его команды.

Части JavaScript

Из каких частей состоит язык JavaScript

В этой статье рассмотрим из каких частей состоит язык JavaScript.

Переменная:

Создание переменной

`var имяПеременной;` // объявление переменной,

Присвоение переменной значения

`имяПеременной = значение;` // инициализация переменной,

Тип данных, он зависит от значения, может быть:

`number` (число),

`string` (строка),

`boolean` (логический тип данных),

`object` (объект),

`undefined` (неопределённый),

`null` (отсутствующий).

Операции, бывают следующих видов:

Присваивания `=`,

Арифметические `+, -, *, /, %,`

Сравнения `==, >, >= <, <=, !=, !==, ===,`

Логические `&&, ||, !, !!`.

Операторы ветвления, бывают следующих видов:

`if`,

`if else`,

`if else if`,

`switch`

Циклы, бывают следующих видов:

`for`,

`while`,

`do while`,

`foreach`

Пользовательская функция, схема создания:

`function имяФункция() {`

```
}
```

Массив, схема создания:

```
var имяМассива;

имяМассива = [ ];
```

Объект, схема создания:

```
var имяОбъекта;

имяОбъекта = { };
```

Ключевые слова JavaScript

Какие ключевые слова существуют в JavaScript?

Ключевые слова в JavaScript — это слова, которые существуют в ядре языка JavaScript и встроены в его синтаксис, например слово `var`.

В качестве имён идентификаторов *(переменных, функций, объектов, массивов и т.д.)* **нельзя**использовать имена ключевых слов.

Список ключевых слов JavaScript

- `arguments`,
- `break`,
- `case`,
- `catch`,
- `continue`,
- `default`,
- `delete`,
- `do`,
- `else`,
- `eval`,
- `finaly`,
- `for`,
- `function`,
- `if`,
- `in`,
- `instanceof`,
- `new`,
- `null`,
- `return`,
- `switch`,
- `this`,
- `throw`,
- `try`,

- `typeof`,
- `var`,
- `void`,
- `while`,
- `with`.

Зарезервированные слова JavaScript

Какие зарезервированные слова существуют в JavaScript?

Зарезервированные слова в JavaScript — это слова, которые пока еще не существуют в ядре языка JavaScript и не встроены в его синтаксис, но в будущем, эти слова могут быть внедрены в ядро JavaScript, например слово `abstract`.

В качестве имён идентификаторов *(переменных, функций, объектов, массивов и т.д.)* **не рекомендуется** использовать названия зарезервированных слов.

Список зарезервированных слов JavaScript

- `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`.

Идентификаторы JavaScript

Что такое идентификатор в JavaScript?

Идентификатор — это уникальное имя предмета, позволяющее отличать его от других предметов.

Идентификатор в JavaScript — это уникальное имя:

- переменной,
- функции,
- объекта,
- массива,
- меток,

и других элементов синтаксиса JavaScript, где пользователь самостоятельно назначает имена.

Правила задания имён идентификаторов в JavaScript

Имена **идентификаторам в JavaScript**, вы придумываете самостоятельно.

Правила задания имён идентификаторов:

- имя идентификатора может состоять из любых цифр и букв английского алфавита,
- в имени идентификатора, могут использоваться символы доллара `$` и подчёркивания `_`
- имя идентификатора, не может начинаться с цифры,
- в качестве имён идентификаторов *нельзя/не рекомендуется* использовать ключевые и зарезервированные слова JavaScript,
- имена идентификаторов регистрозависимы, это значит что «Name» и «name», это разные идентификаторы.

Приведём примеры правильных и не правильных имён идентификаторов:

Правильно	Не правильно
name	na me
$name	na*me
_name	na-me
name5	5name
NAME	+name+

Создавайте осмысленные имена идентификаторов

При создании имен идентификаторов, обычно им дают осмысленные названия, например `cvetAvto`, а не peremennaya1. Для вашего удобства, старайтесь создавать имена идентификаторов, описывающие данные, которые они представляют.

Например, если переменная хранит данные о цвете автомобиля, то её имя может быть таким `cvetAvto` или `colorAuto`, если функция изменяет цвет автомобиля, то её имя может быть таким `izmenitCvetAvto` или `changeColorAuto`.

Имена переменных обычно представляют собой **имя существительное**, а имена функций представляют собой **глагол**.

Опытные программисты обычно используют верблюжий стиль *(camelCase)* написания имён идентификаторов, когда каждое последующее слово начинается с прописной *(заглавной)*буквы **cvetAvto**. Если вам не нравится *camelCase*, то вы можете между словами ставить знак подчёркивания cvet_avto

Комментарии JavaScript

Какие существуют комментарии в JavaScript

В языке JavaScript существуют два вида **комментариев**:

- однострочные `//` и
- многострочные `/* */`

Комментарии JavaScript располагают внутри кода JavaScript, между тегами `<script></script>` или в файле `.js`

Однострочный комментарий

Однострочный комментарий, начинается с двойного слеша `//` и действует на всю строку, после слеша.

Пример использования однострочного комментария *(комментарии выделены красным цветом)*:

```
<script>
 // Это однострочный комментарий
 JavaScript-код;

 JavaScript-код; // Это однострочный комментарий
</script>
```

Многострочный комментарий

Многострочный комментарий начинается с символов `/*` и заканчивается символами `*/`

Многострочный комментарий может располагаться на одной строке или на нескольких. Интерпретатор JavaScript, пропускает символы `/* */` и не обрабатывает их, как и то, что расположено между ними.

Пример использования многострочного комментария:

```
<script>
/*
Это многострочный комментарий
который можно расположить на
нескольких строках
*/
```

```
/* А можно и на одной */


JavaScript-код;
</script>
```

Для чего используются комментарии в JavaScript

Использование комментариев в языке JavaScript, необходимо в двух случаях:

1. Для описания сложной программы, когда вы записываете что, делает та или иная часть данного кода, потом вернувшись к нему через некоторое время, вы сможете вспомнить, что делали и почему
2. Для отладки программы, например, комментариями можно закрывать тот или иной участок кода и в итоге выяснить откуда начинается возникновение ошибки в коде JavaScript