JavaScript

История появления

Задача — обеспечить динамическое изменение веб-страниц на стороне клиента, реагировать на действия пользователя (проверять пользовательский ввод, показывать анимацию и др.) в окне браузера. Язык должен походить на Java и быть достаточно простым для изучения.

Первые шаги — Cmm, CEnvi, Espresso Pages.

Компания Netscape в 1995 г. наняла Брендана Эйха (Brendan Eich), который положил начало языку «Mocha». Язык впоследствии был доработан и переименован в «LiveScript», а позже (4 декабря 1995 года) — в «JavaScript».

Стандартизирован Sun и ECMA, оформлен как ECMAScript, где определены зарезервированные слова, операторы, типы данных и др. Непосредственно название «JavaScript» является торговым знаком компании «Oracle Corporation».

Основные сведения

Классификация — высокоуровневый, сценарный, прототипно-ориентированный, интерпретируемый.

Тип файлов — `.js`

Архитектурные черты — слабая динамическая типизация, прототипы, автоматическое управление памятью, наличие функциональных элементов(функции как объекты первого класса, анонимные ф-ции, замыкания и др.).

Область применения — веб-приложения, мобильные приложения, прикладное ПО, обучение программированию.

Подробнее

Брендан Эйх (4 июля 1961, Питтсбург, Пенсильвания, США) Основатель Mozilla Corp., создатель JavaScript

Логотип JavaScript

Прототип — самостоятельный объект, полученный в результате клонирования другого, наследующий характеристики родителя с возможностью их переопределения.

Объекты первого класса — элементы языка программирования, которые могут быть присвоены переменной, переданы в качестве параметра, возвращены как результат и др.

Подключение JS

Существует 3 варианта подключения JavaScript к HTML-разметке:

Напрямую в коде тэгом <script> … </script>

```
<body>
    <script type="text/javascript" language="javascript">
        alert("Hello, World!");
    </script>
</body>
```

Рекомендуется выносить JS в тэг **<head>** при использовании данного способа для сохранения структуры HTML-разметки.

Атрибуты **type** и **language** на текущий момент не обязательны для заполнения.

По ссылке в тэге <script src="…"></script>

```
<head>
    <title>Start Page &mdash; JavaScript Beginner Course 2016</title>
    <meta charset="utf-8">
    <script src="scripts/mainScript.js" defer></script>
</head>
```

Путь может быть как абсолютным (включая домен/адрес), так и относительным.

В одном тэге использовать сразу оба способа подключения (по ссылке + собственно код) нельзя.

defer означает, что скрипты будет выполнены после обработки всей страницы браузером с сохранением относительного порядка, **async** — что скрипты выполнятся асинхронно по мере их загрузки.

Инструменты:

Текстовый редактор с подсветкой синтаксиса и возможностью автодополнения:

Sublime Text 3
Малый размер на диске, высокая скорость работы, разнообразие расширений

Adobe Brackets
Быстрое редактирование стилей, функция живого отображения изменений

IntelliJ IDEA
Огромное количество интеграций, высокие требования к ресурсам ПК, условно-бесплатное распространение

… а также любой современный браузер (Google Chrome, Mozilla Firefox, Apple Safari, Microsoft Edge и др.)

В атрибутах тэга (onclick, onmouseover и др.)

```
<h1 onclick="this.style.color = 'red';">
    Hi there! Let's start to learn JavaScript.
</h1>
```

Чаще всего такой способ подключения используется для назначения обработчиков событий, где `onclick` — обработчик нажатия ЛКМ, `this` указывает на объект, для которого событие произошло.

Структура кода

Рассмотрим, из чего состоит .js-файл:

```
"use strict"; – (1)
alert("Hello from JavaScript!"); – (2)
// Basic output into browser console – (3)
```

1) Директива, указывающая на строгое следование последним стандартам JS. В ES5 были внесены важные изменения и исправления ошибок в дизайне, однако появились проблемы с поддержкой программ, написанных на более старых версиях языка. По умолчанию данные нововведения отключены, `use strict` используется для их активации.
2) Инструкция для выполнения. Может содержать объявления переменных, определения функций, операторы и др. Инструкций может быть несколько, рекомендуется разделять их запятыми, но это не обязательно. Для вложенных инструкций рекомендуется соблюдать отступы.
3) Комментарий (однострочный). Многострочный комментарий — `/* … */`.

Переменные

Инициализация, копирование, доступ

Переменные — важные инструменты для работы с данными, состоят из имени + области памяти, выделяемой для неё. Объявление (создание) переменной:

```
var admin, name = "Alex";
var COLOR_ORANGE = "#FF7F00";
admin = name;
console.log(admin);
```

Можно объявлять несколько переменных в одном блоке `var`, однако инициализация осуществляется только напрямую. Константы обозначаются буквами в верхнем регистре и служат для упрощения написания кода.

Ключевые слова (`for`, `while`, `return` и др.), имена переменных и функций чувствительны к регистру.

В строке (3) осуществляется копирование значения одной переменной в другую, т. е. реализуется доступ по значению.

Область видимости (scope) — часть программы, где данная переменная определена. Существуют глобальные (видны во всей программе) и локальные (видны в ограниченной телом функции области) переменные.

```
var scope = "global";
function scopeTest() {
    console.log(scope); // "undefined"
    var scope = "local";
    console.log(scope); // "local"
}
```

`undefined` выведется в результате «подъема»: локальные переменные в функциях объявляются в первую очередь, а инициализируются в порядке следования.

`let` и `const`

Начиная с ECMAScript 2015 появились новые способы задания переменных:

`let` позволяет сузить область видимости до блока `{...}`, получать доступ к переменным только после объявления, запрещает переинициализацию и упрощает работу с циклами.

`const` запрещает переопределение значения переменной (но не свойств внутри неё).

Функции

Объявление, параметры, возврат значения

Основная задача функции — избежать повторения кода (DRY-принцип). Объявление функции:

```
function printValueAndType (value) {
    console.log(value + " (" + typeof value + ")");
}
```

Классическое определение функции (Functional Declaration) = ключевое слово `function` + имя функции + `(`[список параметров]`)` + `{`тело функции`}`. Создается до выполнения 1й строчки кода.

Функция имеет доступ как к локальным, так и к глобальным переменным, причем как на чтение, так и на запись:

```
var userName = 'Paul McCartney';
alert(greetPaulFromCountry('Japan'));

function greetPaulFromCountry(country) {
    console.log('Hi, here\'s ' + userName);
    userName = 'Ringo Starr'; // overwriting the global name
    return 'Hello from ' + country + ', ' + userName; // returning into the
alert
}
```

Функцию можно вызвать с любым количеством параметров. При передаче значений в функцию изменяются только их локальные копии.

`return` завершает выполнение функции. Если он вызван без параметров, функция вернет `undefined`.

Функциональные выражения

Функцию можно присваивать переменной (привет, Erlang!):

```
var countParticipants = function () {
    return arguments[0].length;
}

var makeBand = function () {
    if (countParticipants(arguments) >= 3)
        console.log('Greetings to Green Day!');
}

makeBand('Billie Joe', 'Mike', 'Tre');
```

Functional Expression можно использовать практически в любом месте программы (например, в блоке `if`).

Можно обрабатывать любое количество входных параметров функции используя псевдомассив `arguments[]`, В ES5 в него попадают все переданные аргументы.

Типы данных

Разновидности, определение типа typeof

`number` — хранение целых и дробных чисел, `Infinity` и `NaN` (ошибка вычислений)

```
var firstNumber = 6, secondNumber = 14.997;
console.log(/*typeof (*/firstNumber + secondNumber/*)*/);
console.log(/*typeof (*/secondNumber / 0/*)*/); // Infinity
console.log(/*typeof (*/"A" * firstNumber/*)*/); // NaN
```

`string` — строка произвольной длины в Юникоде (UTF-16, размер в байтах: 2 · N)

```
var testString = "Luke, I am your father!";
console.log(/*typeof (*/testString/*)*/);
```

`null` — нулевой указатель, ссылка на несуществующий объект

```
var emptyVariable = null;
console.log(/*typeof (*/emptyVariable/*)*/); // design error
```

`boolean` — логический тип, `true` или `false`

```
var equalityCheckResult = 1 < 2;
console.log(/*typeof (*/equalityCheckResult/*)*/);
```

`object` — абстракция над ассоциативным массивом, неупорядоченная коллекция именованных значений

```
var person = {
    name: "Face",
    age: 20
};
console.log(/*typeof (*/person/*)*/);
```

Массивы

Данные можно хранить в виде нумерованного массива:

```
var testArray = ['Coca-Cola', {shop: 'Lenta'},
'Dr. Pepper', 1, true, NaN];
console.log('Array: ' + testArray +
    " | Length: " + testArray.length +
    " | Access to data: " + testArray[1].shop);
```

Массив — разновидность объекта, передается в функции по ссылке. Используется для работы с непрерывным набором нумерованных данных. Основные методы:

1) `pop` и `push` — добавление и удаление элементов в конец массива
2) `shift` и `unshift` — добавление и удаление из начала массива. Работает медленнее, чем (1), т. к. необходимо перемещать элементы массива на новые значения.
3) `length` — «длина» массива, индекс последнего элемента + 1.

Операторы сравнения

Сравнение чисел, строк, данных различных типов

Основные операторы сравнения заимствованы из математики:

```
(2 > -5); (5 <= 10); (101 == 101); (666 != 777); (true > false)
```

Строки сравниваются лексикографически: поочередно сравниваются численные коды букв, любая буква больше отсутствия буквы, строчная буква больше прописной (особенность Unicode, см. Таблицу символов):

```
("Reebok" < "Vans"); ("Rock" < "Rocksmith"); ("usa" > "USA");
```

При сравнении значений разных типов интерпретатор автоматически осуществляет их преобразование (см. стр. 8):

```
('2' > 1); // true, compares as 2 > 1
('01' == 1); // true, compares as 1 == 1
(false == 0); // true, false becomes 0
```

В JavaScript сравнение часто приводит к неожиданным результатам:

```
('' == false);

(null == 0); /* false */ (null > 0); // false again
(null >= 0); // true?

(undefined == null); // true
(undefined > 0); /* false */ (undefined < 0); /* false */ (undefined == 0);
// false
```

`null` приводится к 0 при сравнении, но при проверке на равенство ведет себя как равный только `undefined`. `undefined` приводится к `NaN`, но `null != NaN`.

Строгое равенство

Строгая форма проверки на равенство исключает приведение типов

```
(0 === false); // false
```

Преобразование примитивных типов

Строковое

Происходит когда вызывается `alert`, `console.log` (внутри — скрытый метод `object.toString()`), напрямую вызывается конструктор `String()` или используется бинарный оператор `+` со строкой:

```
var x = 10, y = -3, firstString = "Hello"; console.log(x); // '10'
console.log(firstString + ", " + y + "!"); // 'Hello, -3!'
console.log(new String(null)); // object
```

Числовое

Данные приводятся к числу в математических выражениях и операциях, при сравнении чего-либо с числом, при использование унарного `+` перед строкой, а также при вызове конструктора `Number()`:

```
console.log(typeof + "-9"); // converts string to number
console.log(+ firstString); // NaN
console.log(new Number("007")); // object
console.log(Math.sqrt("64")); // 8
```

`undefined` приводится к `NaN`, `null` к `0`, `true`/`false` к `1`/`0`. Строки очищаются от пробелов и незначащих нулей, и если `length != 0` то либо `NaN`, либо число, иначе `0`.

Логическое

Происходит в логическом контексте (в блоке условия), все интуитивно пустые значения (`0`, пустая строка, `null`, `undefined`, `NaN`) == `false`, все остальные == `true`. Важен контекст!

```
alert( 0 == "\n0\n" ); // 0 == 0
if ("\n0\n") {...} // '0' == true
```

Блок условия

Стандартный условный оператор (тернарный)

Используется для реализации ситуаций логического выбора, выполняет различные инструкции в зависимости от истинности/ложности условия:

```
if (favouriteLanguage != "JavaScript") {
    sayGoodbye();
    return false;
} else /* [if (...)] */ {
    clapYourHands();
    return true;
};
```

Фигурные скобки можно опустить, если в блоке только 1 инструкция. Допускается также сокращенная запись:

```
age > 18 ? welcomeToTheClub(); : howAboutNo();
```

Конструкция `switch` предоставляет наглядный способ сравнения выражения с несколькими вариантами (того же типа данных):

```
switch(x) {
  case 'value1':  // if (x === 'value1')
    ...
    [break]
  default:
    ...
    [break]
}
```

Блок `default` выполняется в случае несоответствия ни одному из вариантов. В случае отсутствия директивы `break` выполнятся все нижестоящие инструкции, проверки проигнорируются.

Циклы

Цикл используется для многократного повторения одинаковых действий. Повторение == итерация.

Цикл for

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

Выполнение цикла:

1) устанавливается начальное значение счетчика;
2) проверяется условие выхода;
3) выполняется тело цикла;
4) счетчик изменяется на величину шага цикла (порядок 3 и 4 шагов может меняться в зависимости от формы инкремента);
5) переход на шаг 2.

Части `for` можно пропустить, получив аналог `while`:

```
for (var i = 0; i < 3; ;) {...} // while (i < 3) {...}
```

Цикл while

```
while (i < 3) {
  alert( i );
  i++;
}
```

Условие выхода проверяется каждый раз перед выполнением тела цикла, в отличие от цикла `do` ... `while`, где реализована пост-проверка.

Инкрементирование

Инкремент можно применять только к переменной. Префиксная форма:

```
var i = 1;
var a = ++i;
alert(a); // 2
```

Сначала увеличивается значение переменной, затем выполняется присваивание (оператор =), в отличие от постфиксной формы:

```
var i = 1;
var a = i++;
alert(a); // 1
```

Аналогично работает и декремент.

Оператор запятая вычисляет все указанные выражения, возвращая последнее:

```
for (a = 1, b = 3, c = a * b; a < 10; a++) {...}
```

Директива `break` завершает выполнение цикла, `continue` завершает текущую итерацию.