JAVASCRIPT.

ЛИТЕРАТУРА.

Флэнаган Дэвид. JavaScript. Подробное руководство. 6-е изд. – Пер. с англ. – СПб: Символ-Плюс, 2012. – 1080с., ил.:

Флэнаган Дэвид. JavaScript. Подробное руководство. 5-е изд.– Пер. с англ. – СПб: Символ-Плюс, 2008. – 992 с., ил.:

Закас Николас. *JavaScript для профессиональных веб-разработчиков* / [Пер. с англ. А. Лютича]. – СПб.: Питер, 2015. – 960с.: ил. – (Серия «Для профессионалов»).

Резинг Джон, Бибо Беэр, Марас Иосип. *Секреты JavaScript ниндзя*. – Пер. с англ. – М.: ООО «И.Д. Вильямс», 2017. – 544 с.: ил.

Резинг Джон, Бибо Беэр. *Секреты JavaScript ниндзя*. – Пер. с англ. – М.: ООО «И.Д. Вильямс», 2013. – 416 с.: ил.

ИНТЕРНЕТ-РЕСУРСЫ.

www.ecma-international.org – официальный сайт ECMA

www.ecma-international.org/publications/standards/Ecma-262.htm - официальная спецификация ECMA (последняя ECMA10, июнь 2019г).

es5.javascript.ru – перевод спецификации EcmaScript 5 с аннотациями. Перевод ресурса https://github.com/iliakan/es5.

es5.github.com – описание ECMAScript 5.1 (от 2015-08-11)

es6-features.org – особенности ECMA6

https://dom.spec.whatwg.org/ - DOM Living Standard, последнее обновление от 23 апреля 2020 г.

https://www.w3.org/TR/html52/semantics-scripting.html#the-script-element - HTML 5.2, W3C Recommendation, 14 декабря 2017 г.

developer.mozilla.org/ru/docs/Web/JavaScript

msdn.microsoft.com

learn.javascript.ru

ИСТОРИЯ

1995г – Брендан Айк (Netscape) + Sun Microsystems = Mocha (LiveScript) для Netscape 2.

LiveScript ⇒ JavaScript

1996г – JavaScript 1.0 → Netscape 3

Microsoft – Internet Explorer 3 → JScript

1997г – JavaScript 1.1 → Европейская ассоциация производителей ВТ

(European Computer Manufacturers Association, *Ecma*)

Технический комитет №39 (TC39)
=
Netscape + Sun + Microsoft + Borland + NOMBAS + др.
⇓
ECMA-262

1998г – Международная организация по стандартизации
(International Organization for Standardization, ISO)
+
Международная электротехническая комиссия
(lnternational Electrotechnical Commission, IEC)
=
ISO/IEC-16262

В настоящее время: ECMA – ассоциация, деятельность которой посвящена стандартизации информационных и коммуникационных технологий.

Реализации ECMAScript: JavaScript, JScript, ActionScript, JScript .NET, QtScript

РЕДАКЦИИ ECMAScript.

Версия ECMAScript = ***редакция*** ECMAScript.

Редак-ция	**Дата публикации**	**Новшества**
1	Июнь, 1997	Похож на JavaScript 1.1 от Netscape
2	Июнь, 1998	Обновление для согласования с ISO/IEC 16262 и не содержала изменений
3	Декабрь, 1999	Добавлены: – регулярные выражения, – некоторые управляющие инструкции, – обработка исключений с помощью блока try/catch, Улучшены алгоритмы: – обработки строк, – определения ошибок и вывода чисел.
4	Не опублико-ван	Спецификация комитета Technical Committee (TC39) включала: – строго типизированные переменные, – новые инструкции и структуры данных, – полноценные классы, – классическое наследование и новые способы взаимодействия с данными. Альтернативное предложение TC39 – спецификация ECMAScript 3.1. В результате ES3.1 «приостановил» разработку ES4!

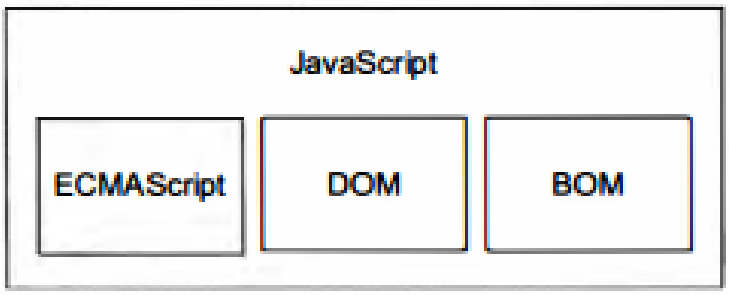
5	Декабрь, 2009	Спецификация ECMAScript 3.1 = ES5: – проясняются неоднозначные места третьей редакции – вводится новая функциональность: объект JSON, Object.create(), Object.defineProperty(), аксессоры (getters и setters) – методы наследования и расширенного определения свойств – многострочные строковые литералы – строгий режим
5.1	Июнь, 2011	Редакция 5.1 ECMAScript полностью приведена в соответствии со стандартом ISO/IEC 16262:2011.
6	Июнь, 2015	ECMAScript 6 (ES6) = ECMAScript 2015 (ES2015) Добавлены: – классы, – модули (в «строгом режиме»), – цикл for/of, – типированные массивы, – коллекции (maps, sets и weak maps), – замыкания – и др.
7	Июнь, 2016	The 7th Edition = ECMAScript 2016, Добавлены: – оператор **, – Array.prototype.includes, и др.
8	Июнь, 2017	Планировалось: – «улучшить» работу с числами и математическими выражениями, – «улучшить» классы и instance-свойства, – работа с потоками – перегрузка операторов, – записи и кортежи и др. Появилось: – поддержка асинхронности (async/await); – «висячие» запятые в параметрах функций (возможность ставить запятые в конце списка аргументов функций); – два новых метода для работы со строками: padStart() и padEnd().добавлена функция Object.getOwnPropertyDescriptors() – возвращает массив с дескрипторами всех собственных свойств объекта; – разделение памяти (тип SharedArrayBuffer) и объект Atomics.
9	Июнь, 2018	Новое: Из https://webformyself.com/chto-novogo-v-es2018-javascript – Асинхронная итерация – Promise.finally() – Свойства Rest/Spread (преобразование последовательности чисел в/из массив) – Именованные группы захвата регулярных выражений – Утверждения регулярных выражений lookbehind – Флаг регулярных выражений s (dotAll) – Переход к свойствам Unicode через регулярные выражения – Изменения в литералах шаблонов
10	июнь, 2019	– Необязательный аргумент у catch – Доступ к описанию символьной ссылки

		– Строки EcmaScript совместимые с JSON – Доработка прототипного метода .toString() – Создание объекта методом Object.fromEntries() – Одномерные массивы с .flat() и .flatMap() *ES*10 — всё ещё черновик. *ES*9 — актуальная версия спецификации. (с https://habr.com/ru/post/437806/)
	ES.NEXT	Любой будущий стандарт

РЕАЛИЗАЦИИ JAVASCRIPT

Полная реализация JavaScript сосоит из:

- ядро (*ECMAScript*);
- объектная модель документа (Document Object Model, *DOM*);
- объектная модель браузера (Browser Object Model, *BOM*).



1. *ECMAScript.*

ECMAScript, определённый в ECMA-262, **НЕ** связан с веб-браузерами!!!

Стандарт ECMA-262 определяет **основу** для создания языков сценариев.

Веб-браузеры – это **среда выполнения**, в которых работает ECMAScript-реализация.

Среда выполнения содержит:

– базовую ECMAScript-реализацию
– расширения, разработанные для взаимодействия с самой средой.

Примеры других сред выполнения:
– NodeJS (серверная JavaScript-платформа)
– Adobe Flash (Adobe ActionScript).

Стандарт ECMA-262 на базовом уровне **определяет**:
– синтаксис;
– типы;
– инструкции;
– ключевые слова;
– зарезервированные слова;
– операторы;
– объекты.

!!! ECMAScript – это просто описание языка, а JavaScript – это реализация ECMAScript

2. *Объектная модель документа*

Document Object Model, DOM

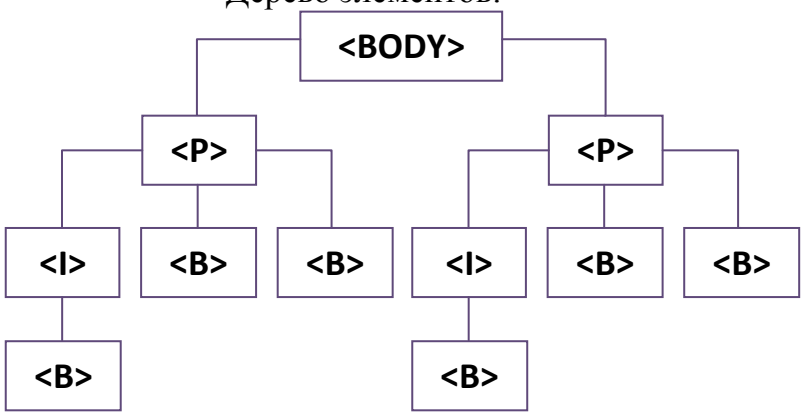
прикладной программный интерфейс (Application Programming Interface, API)
для XML расширенный на HTML.

Объект – html-блок (<P>, <IMG>, <BODY>, <I>, …).
Дерево объектов: корень – элемент <HTML>.

Пример 1.
HTML-разметка: DOM Level 2

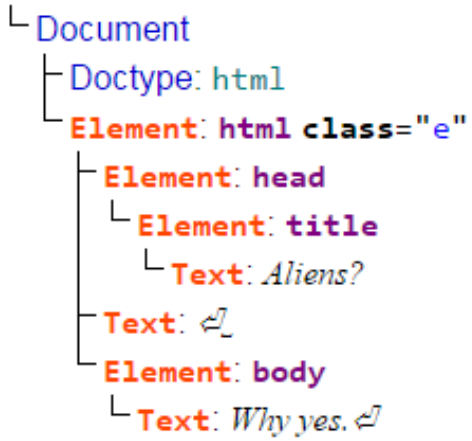
```
<BODY>
  <P>
    <I>
      <B>Фрукты:</B>
    </I>
    <B>Груши</B>
    <B>Яблоки</b>
  <P>
    <I>
      <B>Овощи:</B>
    </I>
    <B>Репа</B>
    <B>Свёкла</B>
  </P>
</BODY>
```

Дерево элементов:



Пример 2. DOM Level 4.

```
<!DOCTYPE html>
<html class=e>
 <head><title>Aliens?</title></head>
 <body>Why yes.</body>
</html>
```



Он-лайн преобразование html-документа в DOM-дерево:
http://software.hixie.ch/utilities/js/live-dom-viewer/

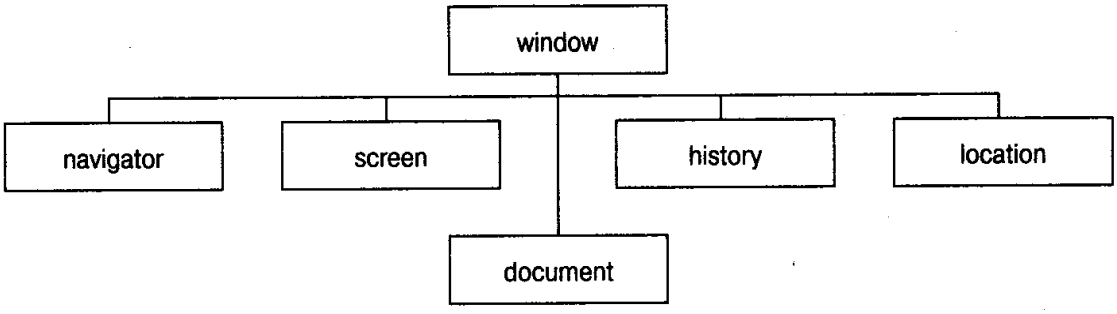
DOM Level 0 – не стандартизированна
DOM Level 1 от 01 октября 1998 - *https://www.w3.org/TR/REC-DOM-Level-1/*
DOM Level 2 от 13 ноября 2000 - *https://www.w3.org/TR/DOM-Level-2-Core/*
DOM Level 3 от 07 апреля 2004 - *https://www.w3.org/TR/DOM-Level-3-Core/*
DOM Level 4 от 19 ноября 2015 - *https://www.w3.org/TR/dom/*
DOM Living Standard, последнее обновление от 23 апреля 2020 г. – *https://dom.spec.whatwg.org/*

3. *Объектная модель браузера*

Brouser Object Model, BOM

BOM регламентирует работу с:

1. окном и фреймами браузера,
2. любым JavaScript-расширением для браузера:
   - отображение всплывающих окон в браузере;
   - возможность перемещать, закрывать и изменять размеры окна браузера;
   - объект `navigator`, предоставляющий сведения о браузере;
   - объект `location`, предоставляющий сведения о странице, загруженной в браузере;
   - объект `screen`, предоставляющий сведения о разрешении экрана;
   - поддержка `cookie`-файлов;
   - настраиваемые объекты (`XMLHttpRequest`, а также `ActiveXObject` в IE).



ПРАВИЛА НАПИСАНИЯ СКРИПТОВ

Правила написания скриптов.

- Тег <SCRIPT>.

```
<SCRIPT [language="Язык скрипта"]
        [src="Файл.js"]
        type="media_тип"
        defer
        async
>
     Текст скрипта
</SCRIPT>
```

Значения атрибута **type**:

```
    text/javascript
    text/ecmascript
    application/ecmascript
    application/javascript
```

www.iana.org/assignments/media-types/media-types.xhtml

```
<META http-equiv="Content-Script-Type" content="text/javascript">
```

- Внешний файл.

```
<SCRIPT src="menu.js">
  // пробельные символы
  // или комментариями
</SCRIPT>
```

- Атрибуты HTML-элементов.

```
<button onclick="alert('Hello!');"> Привет </button>
```

- Псевдо протокол javascript:.

комментарии – `/* */`, а не `//`.

Пример.

```
javascript:var now = new Date(); "<h1>Время:</h1>" + now;
```

(возвращаемое значение – строка с html-разметкой)

```
javascript:alert("Hello World!")
```

(возвращаемое значение – *undefined*)

```
javascript:window.open("about:blank"); void 0;
```

(явное указание: «Не возвращать никакое значение»)

НО!

```
javascript:window.open("about:blank");
```

(возвращаемое значение – `[object Window]`)

Пример 1.

```
<a href='javascript:
  var e = "", r = ""; /* Вычисляемое выражение и результат */
  do {
    /* Отобразить выражение и результат, а затем
       запросить новое выражение */
    e = prompt("Выражение: " + e + "\n" + r + "\n", e);
    try
      /* Попробовать вычислить выражение */
      { r = "Результат: " + eval(e); }
    catch(ex)
      /* Или запомнить ошибку */
      { r = ex; }
  } while(e);
  /* продолжать, пока не будет введено пустое выражение, */
  /* или щелкнуть на кнопке отмены */
  /* Предотвратить замену текущего документа */
  void 0;
'>
  Вычислить значение выражения
</a>
```

Пример 2. Закладка–bookmarklet

Выполнение JavaScript-программ.

JavaScript-программа =

встроенные сценарии +
обработчики событий в разметке HTML +
URL-адреса javascript: +
внешние сценарии JavaScript (атрибуты src тегов <SCRIPT>).
⇓
Window, *Document*.

window.

Два этапа выполнения сценариев:

- загрузка содержимого документа, выполнение кода в элементах <SCRIPT>
- после загрузки документа и выполнения сценариев

Особенность JavaScript!

Однопоточная модель выполнения

Пример 1. Синхронное (блокирующее) выполнение сценариев.
Ультрасовременный JavaScript-код в 1996 г.

```
<html>
  <head>
    <title> Сегодняшняя дата </title>
    <script language="JavaScript">
      function print_todays_date() {
        var d = new Date();
        document.write(d.toLocaleString());
      }
    </script>
  </head>
  <body>
    Дата и время:<br>
    <script language="JavaScript">
      print_todays_date();
    </script>
////////
  </body>
</html>
```

Пример 2. Отложенное и асинхронное выполнение сценариев.

```
<script defer src= "defer.js"> </script>
<script async src= "async.js"> </script>
???????????
```

Пример 3. Эмуляция асинхронного выполнения сценариев.

```
function loadasync(url) {
  // Отыскать <head>
  var head = document.getElementsByTagName("head")[0];
  // Создать элемент <script>
  var s = document.createElement("script");
  // Установить атрибут src
  s.src = url;
  // Добавить <script> в <head>
  head.appendChild(s);
}
```

Пример 4. Событийная модель выполнения сценариев.
DOM Level 0

```
window.onload = function() { ... };
document.getElementByld('button').onclick = function() { ... };

function handleResponse() { ... };
request.onreadystatechange = handleResponse;

function modifyText() { ... }
var el = document.getElementById("myDIV");
el.addEventListener("click", modifyText, false);
e1.addEventListener('dblclick', modifyText, false);
e1.addEventListener('dblclick', anotherModify, false);
...
e1.removeEventListener('click', modifyText, false);
e1.removeEventListener('dblclick', modifyText, false);

<div id = “myDIV” onclick=”код”> … </div>
```

Последовательность выполнения клиентских сценариев.

1.
   - Начинается разбор веб-страницы
   - Создается объект *Document* (объекты *Element* и *TextNode*)
   - ***document.readyState = loading***
2.
   - Добавление элементов <SCRIPT ~~async~~ ~~defer~~> в документ.
   - Их синхронное выполнение (`document.write()`).
3.
   - Загрузка и выполение сценариев <SCRIPT `async`>.
   - Продолжение разбора документа (~~`document.write()`~~).
   - Загрузка сценариев <SCRIPT `defer`> (~~`document.write()`~~)
4.
   - ***document.readyState = interactive*** – завершён разбор документа
5.
   - Выполнение сценариев <SCRIPT `defer`> (`document.write()`).
   - Возможно выполнение сценариев `async`.
6.
   - Событие «`DOMContentLoaded`» в объекте `Document`.
   - Возможно выполнение сценариев `async`.
7.
   - Синтаксический анализ документа завершен
   - Возможно ещё загружается дополнительное содержимое (`img, css, iframe, frame`)
8.
   - Всё содержимое загружено.
   - Все сценарии выполнены.
   - Событие *load* в объекте *Window*.
   - ***document.readyState = complete***

I этап

9.
   - «Работа» обработчиков событий.

II этап

Особенности идеализированной модели:

- Событие «`load`» реализовано везде.
- `async` – стандарт HTML5.
- `document.readyState = loading | interactive | complete | ???`
  `onreadystatechange` – связанное событие

Пример 1. Разные состояния загрузки страницы (developer.mozilla.org).

```
switch (document.readyState) {
  case "loading":
    // Страница все еще загружается
    break;
```

```
    case "interactive":
      // Страница уже загружена.
      // Теперь можно получить доступ к DOM объектам.
      var span = document.createElement("span");
      span.textContent = "A <span> element.";
      document.body.appendChild(span);
      break;
    case "complete":
      // Страница загружена вместе
      // с дополнительными ресурсами.
      console.log("The first CSS rule is: " +
         document.styleSheets[0].cssRules[0].cssText);
      break;
  }
```

Пример 2. *readystatechange* как альтернатива событию *DOMContentLoaded*

```
// альтернатива событию DOMContentLoaded
document.onreadystatechange = function () {
  if (document.readyState == "interactive") {
    initApplication();
  }
}
```

Пример 3. *readystatechange* как альтернатива событию *load*

```
// альтернатива событию load
document.onreadystatechange = function () {
  if (document.readyState == "complete") {
    initApplication();
  }
}
```