Московский государственный технический университет
имени Н. Э. Баумана

В. И. Кузнецов

Язык программирования JavaScript

Методические указания к выполнению
лабораторной работы №3 по дисциплине
«Языки интернет программирования»

Редакция от 1.09.2025

Факультет «Информатика и системы управления»
Кафедра «Компьютерные системы и сети»

Кузнецов В.И.

Язык программирования JavaScript: методические указания к выполнению практикума №3 и лабораторной работы № 3 по дисциплине «Языки интернет-программирования» / В. И. Кузнецов.

Это пособие знакомит с языком JavaScript как с универсальным языком программирования общего назначения. Акцент сделан на синтаксисе, принципах работы со значениями и функциями, асинхронности и модульности. Методическое пособие сосредоточено на самом языке: синтаксисе, типах, функциях, коллекциях, прототипах, классах, обработке ошибок и асинхронном программировании.

Для студентов МГТУ им. Н.Э. Баумана, обучающихся по направлению «Информатика и вычислительная техника».

Лабораторная работа №3

Программирование на языке JavaScript

Цель работы – знакомство с языком программирования JavaScript, освоение функционального стиля программирования, получение общих сведений о стандартных функциях языка.

ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

JS (JavaScript) – функциональный сверхвысокоуровневый интерпретируемый язык программирования. Этот язык и его модификации занимают огромную часть современной работки приложений и в основном используется для работки пользовательской части приложений (FrontEnd).

Изначально JavaScript был создан, чтобы «сделать веб-страницы живыми». Программы на этом языке называются скриптами. Они могут встраиваться в HTML и выполняться автоматически при загрузке веб-страницы. Скрипты распространяются и выполняются, как простой текст. Им не нужна специальная подготовка или компиляция для запуска.

Сегодня JavaScript может выполняться не только в браузере, но и на сервере или на любом другом устройстве, которое имеет специальную программу, называющуюся «движком» JavaScript. Сейчас существуют следующие движки, используемые разработчиками отдельно от браузера:

- V8 - Chrome, Opera. Edge и т.д.
- SpiderMonkey - Firefox
- Chakra – IE
- SquirrelFish – Safari
- И так далее

Современный JavaScript – это «безопасный» язык программирования. Он не предоставляет низкоуровневый доступ к памяти или процессору, потому что изначально был создан для браузеров, не требующих этого. Возможности JavaScript сильно зависят от окружения, в котором он работает. Например, Node.js поддерживает функции чтения/записи произвольных файлов, выполнения сетевых запросов и т.д. В браузере для JavaScript доступно всё, что связано с манипулированием веб-страницами, взаимодействием с пользователем и веб-сервером.

Языки надстройки над JavaScript

Современные инструменты делают трансляцию очень быстрой и прозрачной, фактически позволяя разработчикам писать код на другом языке, автоматически преобразуя его в JavaScript «под капотом».

Примеры таких языков:

- CoffeeScript - добавляет «синтаксический сахар» для JavaScript. Он вводит более короткий синтаксис, который позволяет писать чистый и лаконичный код. Обычно такое нравится Ruby-программистам.
- TypeScript - концентрируется на добавлении «строгой типизации» для упрощения разработки и поддержки больших и сложных систем. Разработан Microsoft.
- Flow - тоже добавляет типизацию, но иначе.
- Dart - стоит особняком, потому что имеет собственный движок, работающий вне браузера (например, в мобильных приложениях). Первоначально был предложен Google, как замена JavaScript, но на данный момент необходима его трансляция для запуска так же, как для вышеперечисленных языков.
- Brython - транслирует Python в JavaScript, что позволяет писать приложения на чистом Python без JavaScript.

Стандарт и синтаксис

JavaScript развивается через спецификацию ECMAScript. Синтаксис отдаленно схож с синтаксисом языка Си , однако имеет ряд существенных отличий:

- Возможность работы с объектами, в том числе определение типа и структуры объекта во время выполнения программы.
- Возможность передавать и возвращать функции как параметры, а также присваивать их переменной.
- Наличие механизма автоматического приведения типов.
- Автоматическая сборка мусора.
- Использование анонимных функций.

Переменные

Так как язык программирования JavaScript является не типизированным. В переменных может храниться любой тип данных и даже функции.

Для именования переменных JavaScript существует набор правил:

- Имена переменных чувствительны к регистру (y и Y это две разных переменных)
- Имена переменных должны начинаться с буквы, символа "$" или символа "_"
- Имя переменной может состоять из любых цифр и букв, а также символов "$" и "_"
- В качестве имени переменной нельзя использовать зарезервированные и ключевые слова

Ключевые слова JavaScript :

```
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, yield, let, const, class, try
```

Также в JavaScript есть зарезервированные слова, не являющиеся частью языка, но могущие войти в него в будущем (мы рассматриваем стандарт ECMA-262):

```
enum, export, extends, import, super
```

Также не рекомендуется, а в некоторых случаях и не разрешается использовать в качестве идентификаторов следующие слова:

```
implements, private, public, interface, package, protected, static
```

Переменная объявляется через ключевые слова const для констант, let для стандартного объявления, var для переменных без ограничения видимости блоком или функции (глобальной) и сейчас редко используется. Const как ключевое слово используется для объявления констант.

```
let x = 10;
```

```
const y = 5;

var  z = 15;
```

Типы данных

Числовой тип данных (number) представляет как целочисленные значения, так и числа с плавающей точкой. Существует множество операций для чисел, например, умножение *, деление /, сложение +, вычитание - и так далее. Кроме обычных чисел, существуют так называемые «специальные числовые значения», которые относятся к этому типу данных: Infinity, -Infinity и NaN.

В JavaScript тип number не может безопасно работать с числами, большими, чем (2^{53}-1) (т. е. 9007199254740991) или меньшими, чем -(2^{53}-1) для отрицательных чисел. Если говорить совсем точно, то, технически, тип number может хранить большие целые числа (до $1.7976931348623157 * 10^{308}$), но за пределами безопасного диапазона целых чисел ±(2^{53}-1) будет ошибка точности, так как не все цифры помещаются в фиксированную 64-битную память. Поэтому можно хранить «приблизительное» значение.

Тип BigInt был добавлен в JavaScript, чтобы дать возможность работать с целыми числами произвольной длины. Чтобы создать значение типа BigInt, необходимо добавить n в конец числового литерала.

```
// символ "n" в конце означает, что это BigInt

const bigInt = 1234567890123456789012345678901234567890n;
```

Строка (string) в JavaScript должна быть заключена в кавычки. В JavaScript существует три типа кавычек.

- Двойные кавычки: "Привет".
- Одинарные кавычки: 'Привет'.
- Обратные кавычки: `Привет`.

Двойные или одинарные кавычки являются «простыми», между ними нет разницы в JavaScript. Обратные же кавычки имеют расширенную функциональность. Они позволяют нам встраивать выражения в строку, заключая их в ${…}.

Булевый тип (boolean) может принимать только два значения: true (истина) и false (ложь).

Специальное значение null не относится ни к одному из типов, описанных выше. В JavaScript null не является «ссылкой на несуществующий объект» или «нулевым указателем», как в некоторых других языках. Это просто специальное значение, которое представляет собой «ничего», «пусто» или «значение неизвестно».

Специальное значение undefined также стоит особняком. Оно формирует тип из самого себя так же, как и null. Оно означает, что «значение не было присвоено». Если переменная объявлена, но ей не присвоено никакого значения, то её значением будет undefined.

Тип symbol (символ) используется для создания уникальных идентификаторов в объектах.

Тип object (объект) – особенный. Все остальные типы называются «примитивными», потому что их значениями могут быть только простые значения (будь то строка, или число, или что-то ещё). В объектах же хранят коллекции данных или более сложные структуры.

Объекты же используются для хранения коллекций различных значений и более сложных сущностей. В JavaScript объекты используются очень часто, это одна из основ языка. Объект может быть создан с помощью фигурных скобок {…} с необязательным списком свойств. Свойство – это пара «ключ: значение», где ключ – это строка (также называемая «именем свойства»), а значение может быть чем угодно.

```
let user = new Object(); // синтаксис "конструктор объекта"

let user = {}; // синтаксис "литерал объекта

let user = { // объект

name: "John", // под ключом "name" хранится значение "John"

age: 30 // под ключом "age" хранится значение 30

};
```

Массивы в JS так же являются объектом.

```
let arr = new Array();

let arr = [];

let fruits = ["Яблоко", "Апельсин", "Слива"];
```

В массиве могут храниться элементы любого типа. Например:

```
// разные типы значений

let arr = [ 'Яблоко', { name: 'Джон' }, true, function() {
alert('привет'); } ];

// получить элемент с индексом 1 (объект) и затем показать его свойство

alert( arr[1].name ); // Джон

// получить элемент с индексом 3 (функция) и выполнить её

arr[3](); // привет
```

ОПЕРАЦИИ

Операции в JavaScript условно можно разделить на несколько видов:

- Арифметические операторы
- Операторы присваивания
- Операторы сравнения
- Логические операторы

Все результаты примеров будем рассматривать при исходных данных y = 5.

- Сложение "+"

выражение x = y + 2 даст результат x = 7; y = 5;

- Вычитание "-"

выражение x = y - 2 даст результат x = 3; y = 5;

- Умножение "*"

выражение x = y * 2 даст результат x = 10; y = 5;

- Деление "/"

выражение x = y / 2 даст результат x = 2.5; y = 5;

- Возведение в степень "**"

выражение x = y ** 2 даст результат x = 25; y = 5;

Говоря об операторе сложения "+" необходимо отметить еще две его важные функции. Первая - **сложение строк**. Если использовать его на строковых значениях, то он выполняет операцию их соединения.

Однако, самое интересное начинается, если мы складываем переменные с данными разного типа. В данном случае начинает действовать неявное приведение типов, которое срабатывает по таким правилам:

1. Если хотя бы один из аргументов является строковым типом, то производится попытка привести второй аргумент к строке и соединить их. Эта попытка будет удачной всегда - цифровое значение представляется строкой, null, NaN и Undefined тоже превращаются в строку с названием своего типа данных, соответственно "null", "NaN" и "Undefined".
2. В случае, если ни один из аргументов не является строкой, тогда оба аргумента приводятся к цифровому типу и арифметически складываются. Приведение происходит по тому же принципу, что и явное приведение функцией Number().

Инкремент "++" Эта операция производит увеличение аргумента на единицу, т.е. выражение x++ будет эквивалентно выражению x = x + 1.

Декремент "--" А эта операция производит уменьшение значения переменной на единицу, т.е. x-- эквивалентно x = x - 1. Порядок применения декремента в javascript также имеет значение, например:

Порядок применения инкремента в javascript имеет значение, например:

- выражение x = ++y даст результат x = 6, y = 6, так как вначале увеличивается значение переменной y на единицу и затем переменной x присваивается значение переменной y;

- а выражение x = y++ даст результат x = 5, y = 6, поскольку в данном случае операция инкрементирования (увеличения) происходит ПОСЛЕ того как переменной x присвоили значение из переменной y.

Оператор = это обычный оператор присваивания. Выполнение x = y приведет к x = 5.

Некоторые арифметические операторы можно использовать вместе с оператором присваивания:

- Оператор += Это присваивание со сложением Выполнение x += y приведет к x = 15, эквивалентно x = x + y
- Оператор -= Это присваивание с вычитанием Выполнение x -= y приведет к x = 5, эквивалентно x = x - y
- Оператор *= Присваивание с умножением Выполнение x *= y приведет к x = 50, эквивалентно x = x * y
- Оператор /= Присваивание и деление. Выполнение x /= y приведет к x = 2, эквивалентно x = x / y
- Оператор %= Присваивание с операцией "остаток от деления" Выполнение x %= y приведет к x = 0, эквивалентно x = x % y

В JavaScript имеется два варианта условных операторов: конструкция if ... else и конструкция switch. Первая из них используется для выполнения некоторого кода, если указанное условие истинное.

```
if (условие) {
  // код для выполнения если условие истинно
}
```

В JavaScript также имеется так называемый условный оператор (его также иногда называют тернарный оператор), он присваивает значение переменной на основе некоторого условия. Синтаксис условного оператора выглядит так:

```
имя_переменной = (условие) ? значение1 : значение2;
```

ЦИКЛЫ

В JavaScript мы имеем два вида циклов:

- for - выполнение кода указанное количество раз
- while – выполнение с пред проверкой
- do…while – выполнение с пост проверкой

У цикла for существует разновидность - конструкция **for ... in**
Этой операцией производится перебор всех разрешенных свойств объекта.

```
for (переменная in объект) {
  исполняемый код
}
```

Оператор for...of выполняет цикл обхода итерируемых объектов (включая Array, Map, Set, объект аргументов и подобных), вызывая на каждом шаге итерации операторы для каждого значения из различных свойств объекта.

```
for (переменная of объект) {
  исполняемый код
}
```

Функции

Функции бывают объявлением (function declaration), функциональным выражением и стрелочными. Стрелочные функции не имеют собственного this и подходят для коротких колбэков.

```
function add(a, b) { return a + b; }
const mul = function(a, b) { return a * b; };
const pow = (a, b) => a ** b;
console.log(add(2,3), mul(2,3), pow(2,3));
```

Если в двух словах замыкание — это такая функция, которая была объявлена внутри другой функции. Есть еще одно условие - эта функция должна иметь доступ к переменным функции, внутри которой она была объявлена. Таким образом такая функция (замыкание) имеет доступ к данным внутри себя и внутри родительской функции. Так же внутренняя функция может обращаться не только к переменным, но и к входным параметрам своей внешней функции.

```
   function greetPirate(pirateName) {            // Объявление родительской
функции
     var greeting = "Hello ";

     function checkCaptain() {                   // Объявление замыкания
       if (pirateName != "Jack Sparrow")
         return greeting + pirateName;
       else
         return greeting + "CAPTAIN " + pirateName + "!";
     }

     return checkCaptain();
   }
   console.log(greetPirate("Mad Dog"));            // Выведет в консоль
"Hello Mad Dog"
   console.log(greetPirate("Jack Sparrow"));       // Выведет в консоль
"Hello CAPTAIN Jack Sparrow!"
```

Исключения и обработка ошибок

Используйте throw для генерации ошибок и try/catch/finally для обработки. Объект Error имеет имя и сообщение.

```
function parseJson(s) {
  try {
    return JSON.parse(s);
  } catch (e) {
    throw new Error('Неверный JSON');
  } finally {
    // Опциональная очистка ресурсов
  }
}

try {
  parseJson('{bad json}');
```

```
} catch (e) {
  console.log(e.message); // 'Неверный JSON'
}
```

Асинхронность: таймеры, промисы, async/await

Асинхронное программирование позволяет выполнять операции без блокировки потока. Таймеры планируют задачи, промисы инкапсулируют завершение/ошибку, async/await упрощает работу с ними.

```
const delay = ms => new Promise(resolve => setTimeout(resolve, ms));

async function demoAsync() {
  console.log('Начало');
  await delay(500);
  console.log('Прошло 500 мс');
}
demoAsync();

Promise.resolve(5)
  .then(x => x * 2)
  .then(x => console.log(x))    // 10
  .catch(err => console.error(err))
  .finally(() => console.log('Готово'));
```

Модули ES: import/export

Модули помогают структурировать код. Используйте export для экспорта и import для импорта. В браузере можно подключать скрипт с type="module".

```
// utils.js
export const avg = arr => arr.reduce((a,b)=>a+b,0) / arr.length;
```

```
// app.js

import { avg } from './utils.js';

console.log(avg([10, 20, 30])); // 20
```

Способы ввода и вывода информации в JS

В JavaScript способы ввода и вывода делятся на взаимодействие с пользователем и отображение в браузере. Для ввода используются встроенные функции prompt() и элементы <input>, а для вывода — alert() (окно сообщения), document.write() и element.innerHTML (на страницу), а также методы объекта console (log(), warn(), error()) для вывода в консоль браузера.

Способы вывода в JavaScript

Самый распространенный способ для отладки.

console.log(): выводит текст или объекты в консоль разработчика.

console.warn(), console.error(), console.info(): выводят сообщения с разным стилем (предупреждение, ошибка, информационное).

Вывод в виде сообщения пользователю:

alert(): отображает всплывающее окно с сообщением.

confirm(): отображает окно с сообщением и двумя кнопками: "ОК" и "Отмена", возвращая true или false.

Способы ввода в JavaScript

Ввод с помощью prompt():

Функция отображает диалоговое окно с полем ввода и возвращает введенный пользователем текст или null, если пользователь отменил ввод.

ПОРЯДОК ВЫПОЛНЕНИЯ ЗАДАНИЯ

Выполните три задания. Все решения должны быть самодостаточными и запускаться в консоли или через <script> (type="module" при необходимости).

Задание 1. Разбор CSV-строки

Напишите функцию parseCSV(text), принимающую строку CSV и возвращающую массив объектов. Первая строка содержит заголовки. Учтите экранированные поля в кавычках.

```
// Вход: "name,age\nАлиса,20\nБоб,25"

// Выход: [{name:'Алиса', age:'20'}, {name:'Боб', age:'25'}]

function parseCSV(text) {

  // Ваш код

}
```

Задание 2. Работа с массивом

Реализуйте следующий список функций для работы с массивом чисел:

1. min(arr), возвращающую минимальный элемент,
2. max(arr), возвращающую максимальны элемент,
3. sort(order, arr), возвращающую отсортированный массив, порядок сортировки указывается как true или false в параметре order, где true по возрастанию,
4. find(arr), возвращает true или false при наличии элемента в массиве,
5. replicaRemove(arr), возвращающую массив без повторов.

Предусмотрите проверку введенных элементов по типу. При наличии элемента не числового типа выводить ошибку

Задание 3. Консольный магазин

Реализовать простой консольный магазин. Магазин должен содержать 2 сущности:

- Товар: название, цена
- Корзина: список товаров, количество каждого товара

Меню управления должно быть реализовано через контекстное окно или консоль:

1. Добавить товар
2. Добавить товар в корзину
3. Удалить товар из корзины (по одному товару за раз)
4. Конечная стоимость корзины

Контрольные вопросы

1) В чём различия между var, let и const? Объясните области видимости и поднятие.

2) Чем отличаются == и ===? Приведите примеры неявных преобразований.

3) Что такое замыкание и какие практические задачи оно решает?

4) Как устроено прототипное наследование? Как классы соотносятся с прототипами?

5) Для чего нужны операторы ?? и ?., приведите примеры использования.

6) Объясните принципы работы промисов и различия между then/catch/finally и async/await.

7) Как организовать код в виде модулей? Как подключить модуль в браузере?