5137

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

РЯЗАНСКИЙ ГОСУДАРСТВЕННЫЙ РАДИОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

ВЕБ-ПРОГРАММИРОВАНИЕ

Часть 2

Методические указания к лабораторной работе

Рязань 2017

УДК 004.4

Веб-программирование. Часть 2: методические указания к лабораторной работе / Рязан. гос. радиотехн. ун-т; сост.: А.Н. Сапрыкин, А.М. Гостин. Рязань, 2017. 16 с.

Содержат описание лабораторной работы, используемой в курсах "Веб-программирование", "Интернет-технологии".

Предназначены для студентов очной, очно-заочной и заочной форм обучения направления "Информатика и вычислительная техника".

Библиогр.: 4 назв.

Язык JavaScript, веб-страница

Печатается по решению редакционно-издательского совета Рязанского государственного радиотехнического университета.

Рецензент: кафедра САПР вычислительных средств Рязанского государственного радиотехнического университета (зав. кафедрой В.П. Корячко)

Веб-программирование. Часть 2

Составители: С а п р ы к и н Алексей Николаевич
Г о с т и н Алексей Михайлович

Редактор Н.А. Орлова
Корректор С.В. Макушина
Подписано в печать 27.04.17. Формат бумаги 60x84 1/16.
Бумага писчая. Печать трафаретная. Усл. печ. л. 1,0.
Тираж 50 экз. Заказ
Рязанский государственный радиотехнический университет.
390005, Рязань, ул. Гагарина, 59/1.
Редакционно-издательский центр РГРТУ.

Лабораторная работа № 2
Управляющие конструкции языка JavaScript. Стандартные объекты и функции ядра JavaScript

Цель работы: изучение управляющих конструкций, стандартных объектов и функций языка JavaScript; получение практических навыков их применения для разработки веб-приложений.

1. Введение

Данная работа посвящена изучению управляющих конструкций, стандартных объектов и функций языка JavaScript.

В рамках данной лабораторной работы рассматриваются операторы выбора, циклов, пользовательские функции, стандартные объекты Array, Date, Math и стандартные функции верхнего уровня.

2. Теоретическая часть

2.1. Управляющие конструкции языка JavaScript

1. Условный оператор `if`. Синтаксис оператора:

```
if (условие)
{
  // эти операторы выполняются, если условие верно
  оператор_1;
  …
  оператор_n;
}
else {
  // эти операторы выполняются, если условие ложно
  оператор_a;
  …
  оператор_b;
}
```

Условие для проверки записывается в круглых скобках после служебного слова `if`. После него в фигурных скобках описываются выполняемые операторы. Далее может следовать необязательный блок `else`, выполняемый, когда условие ложно. Число операторов, записываемых в фигурных скобках, неограниченно.

Пример использования условного оператора `if`:

```
var year = prompt('В каком году ,был основан Рязанский радиотехнический институт?', '');
```

```
if (year != 1951) {
  alert( 'Неправильно!' );
}
```

Если необходимо проверить несколько вариантов условия, используется блок `else if`. Пример:

```
var year = prompt('В каком году был основан
Рязанский радиотехнический институт?', '');
if (year < 1951) {
  alert( 'Это слишком рано...' );
} else if (year > 1951) {
  alert( 'Это поздновато...' );
} else {
  alert( 'Да, точно в этом году!' );
}
```

Также существует укороченный вариант условного оператора `if` – условный (тернарный) оператор `?`. Синтаксис оператора:

```
условие ? выражение1 : выражение2
```

Оператор возвращает значение *выражения1*, если условие верно, и значение *выражения2* в противном случае. Пример:

```
var access = (result == password) ? 'Пароль введен
верно!' : 'Ошибка!';
alert(access);
```

2. Оператор выбора `switch`. Синтаксис оператора:

```
switch(x) {
  case 'value1':  // if (x === 'value1')
    ...
    break;
  case 'value2':  // if (x === 'value2')
    ...
    break;
  default:
    ...
    break;
}
```

Переменная `x` проверяется на строгое равенство первому значению `value1`, затем второму `value2` и так далее. Если соответствие установлено, то от соответствующей директивы `case` начинает выполняться оператор `switch`. Он выполняется до ближайшего оператора `break` (или если оператор `break` отсутствует,

то до конца оператора `switch`). Если ни один оператор `case` не совпал, то выполняется вариант `default` (при его наличии).

Пример использования оператора выбора `switch`:

```
var a = 2 + 3;
switch (a) {
  case 4:
    alert( 'Маловато' );
    break;
  case 5:
    alert( 'В точку!' );
    break;
  case 6:
    alert( 'Перебор' );
    break;
  default:
    alert( 'Я таких значений не знаю' );
}
```

3. Цикл со счетчиком `for`. Синтаксис цикла со счетчиком:

```
for (начало; условие; шаг) {
  // ... тело цикла ...
}
```

Пример цикла со счетчиком, который вызывает функцию `alert(i)` для `i` от 0 до 4 включительно (до 5):

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

Любая часть цикла со счетчиком `for` может быть пропущена. Например, можно убрать `начало`. Цикл в примере ниже полностью идентичен приведённому выше:

```
var i = 0;
for (; i < 5; i++) {
  alert( i ); // 0, 1, 2, 3, 4
}
```

Можно убрать и `шаг`:

```
var i = 0;
for (; i < 5;) {
  alert( i );
  // цикл превратился в аналог while (i<5)
}
```

Также можно убрать все параметры цикла со счетчиком, при этом получается бесконечный цикл (при удалении параметров цикла нельзя удалять символы разделителя ‘;’, иначе это приведет к ошибке синтаксиса):

```
for (;;) {
  // будет выполняться вечно
}
```

4. Цикл с предусловием `while`. Синтаксис цикла с предусловием:

```
while (условие) {
  // тело цикла
}
```

Пример цикла с предусловием, который вызывает функцию `alert(i)` для `i` от 0 до 4 включительно (до 5):

```
var i = 0;
while (i < 5) {
  alert( i );
  i++;
}
```

5. Цикл с постусловием `do while`. Синтаксис цикла с постусловием:

```
do {
  // тело цикла
} while (условие);
```

Пример цикла с постусловием, который вызывает функцию `alert(i)` для `i` от 0 до 4 включительно (до 5):

```
var i = 0;
do {
  alert( i );
  i++;
} while (i < 5);
```

6. Метки.

Метка представляет собой оператор с идентификатором, который позволяет ссылаться на какое-то место в программе. Например, метка может быть использована для обозначения цикла.

Синтаксис метки:

```
метка :
  оператор
```

Значение метки может быть любым корректным идентификатором, не являющимся зарезервированным словом. Оператор, указанный после метки, может быть любым выражением.

Пример использования метки для обозначения цикла:

```
markLoop:
while (theMark == true) {
  doSomething();
}
```

7. Прерывание цикла.

Оператор прерывания `break` позволяет прервать выполнение цикла и перейти к следующему за ним выражению.

Оператор `break` может быть использован двумя способами:

- при использовании без метки он прерывает циклы `while`, `do while` и `for` и передает управление на строку за телом цикла;
- при использовании с меткой, он прерывает специально отмеченное выражение.

Пример прерывания цикла:

```
var sum = 0;
while (true) {
  var value = +prompt("Введите число", '');
  if (!value) break; // Прерывание цикла при вводе
пустой строки или отмене ввода
  sum += value;
}
alert( 'Сумма: ' + sum );
```

Пример прерывания метки:

```
outer: for (var i = 0; i < 3; i++) {
  for (var j = 0; j < 3; j++) {
    var input = prompt('Значение в координатах
'+i+','+j, '');
    if (!input) break outer; // Прерывание цикла с
меткой ‘outer’ при вводе пустой строки или отмене ввода
  }
}
alert('Готово!');
```

8. Перезапуск цикла.

Оператор перезапуска `continue` прекращает выполнение текущей итерации цикла. Прерывается не весь цикл, а только текущий шаг. Пример перезапуска цикла:

```
for (var i = 0; i < 10; i++) {
```

```
  // вывод нечётных значений
  if (i % 2 == 0) continue;
  alert(i);
}
```

Оператор `continue` также может быть использован с меткой. В этом случае цикл с меткой завершит текущую итерацию.

2.2. Функции

В JavaScript функцией называется именованная часть программного кода, которая выполняется только при обращении к ней посредством указания ее имени. Функции создаются с помощью ключевого слова `function`. Функция должна быть объявлена перед её использованием. Существует три способа объявить функцию.

1. В декларативном стиле.

Синтаксис объявления функции:

```
function идентификатор(параметры) {
  инструкции
  return выражение
}
```

Синтаксис объявления функции обязательно включает в себя служебное слово `function`, идентификатор, определяющий имя функции, пару круглых скобок и пару фигурных скобок, внутри которых располагаются инструкции. Тело функции может быть пустым, но фигурные скобки должны быть указаны всегда.

Пример объявления функции:

```
function showMessage() {
  alert( 'Привет мир!' );
}
```

Для того чтобы вызвать функцию в нужном месте, необходимо указать ее имя:

```
<script language="JavaScript">showMessage();</script>
```

Второй вариант вызова функции непосредственно в HTML теге:

```
<a href="javascript: showMessage ()">Текст ссылки</a>
```

Обычно функции располагают в секции `<head>`. Такое расположение функций в HTML-документе гарантирует их полную загрузку до того момента, когда их можно будет вызвать из секции `<body>`.

Ниже приведен код HTML страницы, после загрузки которой через три секунды появится указанное сообщение, генерируемое вызовом функции `myMessage()`:

```
<html>
<head>
  <title>Пример вызова функции</title>
  <script>
    function myMessage() {
      alert("Текст сообщения")
    }
  </script>
</head>
<body onload='setTimeout ("myMessage()",3000)'>
  <p>Через три секунды появится сообщение</p>
</body>
<html>
```

2. В функциональном стиле.

Функции могут быть созданы внутри выражения. Такие функции, как правило, анонимны:

```
var square = function(number) {
  return number * number;
}
```

Тем не менее, такие функции все же могут иметь определённое имя. Например:

```
var factorial = function fac(n) {return n<2 ? 1 :
n*fac(n-1)};
print(factorial(3));
```

3. В стиле ООП.

Локальные и внешние переменные

Функция может содержать локальные переменные, объявленные через служебное слово `var`. Такие переменные видны только внутри функции. Например:

```
function showMessage() {
  var message = 'Привет, я - Вася!'; // локальная
переменная
  alert( message );
}
showMessage(); // 'Привет, я - Вася!'
alert( message ); // <-- будет ошибка, т.к.
переменная видна только внутри функции
```

Функция может обратиться к внешней переменной, например:

```
var userName = 'Вася';
function showMessage() {
  var message = 'Привет, я ' + userName;
  alert(message);
}
showMessage(); // Привет, я Вася
```

Доступ к внешней переменной возможен не только для чтения, но и для записи. Переменные, объявленные на уровне всего скрипта, называют «глобальными переменными».

2.3. Массивы с числовыми индексами

Массив – разновидность объекта, которая предназначена для хранения пронумерованных значений и предлагает дополнительные методы для удобного манипулирования ими. Язык JavaScript не имеет встроенного типа данных для создания массивов, поэтому для их создания используется объект `Array` и его методы.

Ни размер JavaScript-массива, ни типы его элементов не являются фиксированными. Поскольку размер массива может увеличиваться и уменьшаться в любое время, то нет гарантии, что массив окажется плотным. Иными словами, при работе с массивом может возникнуть ситуация, когда элемент массива будет пустым и вернёт значение `undefined`.

Создание одномерного массива

Массив в JavaScript можно создать тремя способами:

- `var arr = [элемент1, элемент2...];`
- `var arr = new Array(элемент1, элемент2...);`
- `var arr = new Array(число_элементов);`

Элементы массива нумеруются, начиная с нуля.

Чтобы получить нужный элемент из массива, указывается его номер в квадратных скобках:

```
var fruits = ["Яблоко", "Апельсин", "Слива"];
alert( fruits[0] ); // Яблоко
alert( fruits[1] ); // Апельсин
alert( fruits[2] ); // Слива
```

Элемент массива можно заменить:

```
fruits[2] = 'Груша'; // теперь ["Яблоко",
"Апельсин", "Груша"]
```

Можно добавить новый элемент к массиву:

```
fruits[3] = 'Лимон'; // теперь ["Яблоко",
"Апельсин", "Груша", "Лимон"]
```

Общее число элементов, хранимых в массиве, содержится в его свойстве `length`:

```
var fruits = ["Яблоко", "Апельсин", "Груша"];
alert( fruits.length ); // 3
```

Через метод `alert` можно вывести массив целиком, при этом его элементы будут перечислены через запятую:

```
var fruits = ["Яблоко", "Апельсин", "Груша"];
alert( fruits ); // Яблоко,Апельсин,Груша
```

В массиве может храниться любое число элементов любого типа, в том числе строки, числа или объекты. Например:

```
// набор разнотипных элементов
var arr = [ 1, 'Имя', { name: 'Петя' }, true ];
// получить объект из массива и его свойство
alert( arr[2].name ); // Петя
```

Создание двумерного массива

Массивы в JavaScript могут содержать в качестве элементов другие массивы. Это можно использовать для создания многомерных массивов, например, матриц:

```
var matrix = [
  [1, 2, 3],
  [4, 5, 6],
  [7, 8, 9]
];
alert( matrix[1][1] ); // центральный элемент
```

Перебор элементов массива

Для перебора элементов массива используется цикл:

```
var arr = ["Яблоко", "Апельсин", "Груша"];
for (var i = 0; i < arr.length; i++) {
  alert( arr[i] );
}
```

Удаление элементов массива

Для удаления элементов массива используется метод `delete`:

```
var arr = ["Я", "иду", "домой"];
delete arr[1]; // значение с индексом 1 удалено
// теперь arr = ["Я", undefined, "домой"];
alert( arr[1] ); // undefined
```

2.4. Методы изменения массива

- `Fill()` – заполняет все элементы массива от начального до конечного индексов одним значением. Синтаксис метода:

```
arr.fill(value[, start = 0[, end = this.length]])
```

Метод `fill` может иметь до трёх аргументов: `value` (значение, заполняющее массив), `start` (начальный индекс, указывать который необязательно) и `end` (конечный индекс, указывать который необязательно). Пример:

```
[1, 2, 3].fill(4)        // [4, 4, 4]
[1, 2, 3].fill(4, 1)     // [1, 4, 4]
[1, 2, 3].fill(4, 1, 2) // [1, 4, 3]
```

- `Pop()` – удаляет последний элемент из массива и возвращает его значение. Синтаксис метода:

```
arr.pop()
```

Если массив пуст, метод `pop` возвращает `undefined`. Пример:

```
var fruits = ["Яблоко", "Апельсин", "Груша"];
alert( fruits.pop() ); // удалили "Груша"
alert( fruits ); // Яблоко, Апельсин
```

- `Push()` – добавляет один или более элементов в конец массива и возвращает новую длину массива. Синтаксис метода:

```
arr.push(element1, ..., elementN),
```

где `element1,...,elementN` – элементы, добавляемые в конец массива. Пример:

```
var fruits = ["Яблоко", "Апельсин"];
fruits.push("Груша");
alert( fruits ); // Яблоко, Апельсин, Груша
```

- `Reverse()` – меняет порядок элементов в массиве на обратный. Первый элемент массива становится последним, а последний – первым. Синтаксис метода:

```
arr.reverse()
```

Пример:

```
var arr = [1, 2, 3];
arr.reverse();
alert( arr ); // 3,2,1
```

▪ `Sort()` — сортирует элементы массива и возвращает отсортированный массив. Синтаксис метода:

```
arr.sort([compareFunction])
```

Необязательный параметр `compareFunction` указывает функцию, определяющую порядок сортировки. Если параметр опущен, массив сортируется в соответствии со значениями кодовых точек каждого символа *Unicode*, полученных путём преобразования каждого элемента в строку. Пример:

```
var fruit = ['арбузы', 'бананы', 'Вишня'];
fruit.sort(); // ['Вишня', 'арбузы', 'бананы']
var scores = [1, 2, 10, 21];
scores.sort(); // [1, 10, 2, 21]
```

Пример сортировки с использованием функции сравнения:

```
function compareNumeric(a, b) {
  if (a > b) return 1;
  if (a < b) return -1;
}
var arr = [ 1, 2, 15 ];
arr.sort(compareNumeric);
alert(arr);  // 1, 2, 15
```

▪ `Splice()` – изменяет содержимое массива, удаляя существующие элементы, заменяя и/или добавляя новые.

Синтаксис метода:

```
array.splice(start, deleteCount[, elem1, ...,elemN])
```

Метод `splice()` удаляет `deleteCount` элементов, начиная с номера `start`, а затем вставляет `elem1`, ..., `elemN` на их место. Возвращает массив из удалённых элементов.

2.5. Методы доступа

Эти методы не изменяют массив, а возвращают его в ином представлении.

▪ `Concat()` — возвращает новый массив, состоящий из массива, на котором он был вызван, соединённого с другими массивами и/или значениями, переданными в качестве аргументов.

Синтаксис метода:

```
arr.concat(value1, value2, … valueN)
```

Метод `concat` создаёт новый массив, в который копируются элементы из `arr`, а также `value1, value2, ... valueN`. Метод не изменяет данный массив или любой из массивов, переданных в аргументах, а вместо этого возвращает поверхностную копию, содержащую копии тех элементов, которые были объединены с исходными массивами. Пример:

```
var arr = [1, 2];
var newArr = arr.concat(3, 4);
alert( newArr ); // 1,2,3,4
```

- `Join()` – объединяет все элементы массива в строку.

Синтаксис метода:

```
arr.join(separator)
```

Необязательный параметр `separator` определяет строку, разделяющую элементы массива. Если разделитель опущен, элементы массива разделяются запятой `','`. Если разделитель – пустая строка, элементы массива ничем не разделяются в возвращаемой строке.

Пример:

```
var a = ['Ветер', 'Дождь', 'Огонь'];
a.join();    // 'Ветер,Дождь,Огонь'
a.join('-'); // Ветер-Дождь-Огонь'
```

- `Split()` – превращает строку в массив, разбив ее по разделителю. Синтаксис метода:

```
Arr.split([separator][, limit])
```

Необязательный параметр `separator` указывает символы, используемые в качестве разделителя внутри строки. Необязательный параметр `limit` задает ограничение на количество элементов в массиве. Пример:

```
alert( "a,b,c,d".split(',', 2) ); // a,b
```

- `Slice()` – копирует участок массива в новый массив.

Синтаксис метода:

```
arr.slice([begin[, end]])
```

Метод копирует участок массива, начиная с индекса `begin` и заканчивая индексом `end`, не включая его.

Если индекс отрицательный, `begin` указывает смещение от конца последовательности. Вызов `slice(-2)` извлечёт два последних элемента последовательности.

Если `begin` опущен, `slice()` начинает работать с индекса 0.

Если индекс отрицательный, `end` указывает смещение от конца последовательности. Вызов `slice(2,-1)` извлечёт из последовательности элементы начиная с третьего элемента с начала и заканчивая вторым с конца.

Если `end` опущен, `slice()` извлекает все элементы до конца последовательности. Пример:

```
    var arr = ["Почему", "надо", "учить",
"JavaScript"];
    var arr2 = arr.slice(1, 3); // элементы 1, 2 (не
включая 3)
    alert( arr2 ); // надо, учить
```

2.6. Дата и время

Для работы с датой и временем в JavaScript используются объекты `Date`. Синтаксис создания нового объекта:

```
new Date()
```

Пример создания объекта `Date` с текущей датой и временем:

```
var now = new Date();
alert( now );
new Date(milliseconds)
```

Метод создает объект `Date`, значение которого равно количеству миллисекунд (1/1000 секунды), прошедших с 1 января 1970 года GMT+0.

Параметром конструктора может быть строка, в которой записана нужная дата:

```
date1 = new Date("january 14, 2000, 12:00:00");
```

2.7. Объект Math

Объект `Math` является встроенным объектом, хранящим в своих свойствах и методах различные математические константы и функции. В отличие от других глобальных объектов, объект `Math` не является конструктором. Все свойства и методы объекта `Math` являются статическими. Вы ссылаетесь на константу `π` через `Math.PI` и вызываете функцию синуса через `Math.sin(x)`, где `x` является

аргументом метода. Константы в JavaScript определены с полной точностью действительных чисел.

Свойства: `E`, `LN2`, `LN10`, `LOG2E`, `LOG10E`, `PI` и т.д.

Методы: `abs(x)`, `cos(x)`, `sin(x)`, `exp(x)`, `fround(x)`, `log(x)`, `max([x[,y[,...]]])`, `pox(x,y)`, `random()`, `round(x)` и т.д.

2.8. Стандартные функции верхнего уровня

В JavaScript существуют несколько функций, для вызова которых не надо создавать никакого объекта, они находятся вне иерархии объектов.

Функция `parseFloat(parameter)` анализирует значение переданного ей строкового параметра на соответствие представлению вещественного числа.

Функция `parseInt(parameter, base)` пытается возвратить целое число по основанию, заданному вторым параметром.

Эти функции полезны при анализе введенных пользователем данных в полях формы до их передачи на сервер.

Функции `Number(object)` и `String(object)` преобразуют объект, заданный в качестве его параметра в число или строку.

3. Практическая часть

3.1. Самостоятельное задание

Перед тем как приступить к выполнению самостоятельного задания, внимательно изучите теоретическую часть методических указаний. Создайте новую веб-страницу и добавьте к ней скрипты, соответствующие заданию вашего варианта.

Задание по вариантам

1. Задан одномерный массив вещественных чисел. Напишите функцию, которая определяет число положительных элементов массива.

2. Задан одномерный массив целых чисел. Напишите функцию, которая удаляет в массиве все числа, которые повторяются более двух раз.

3. Задан одномерный массив вещественных чисел. Напишите функцию, находящую сумму отрицательных элементов массива.

4. Задан одномерный массив целых чисел. Напишите функцию, находящую произведение элементов массива с нечетными номерами.

5. Задан одномерный массив вещественных чисел. Напишите функцию, находящую сумму номеров минимального и максимального элементов.

6. Задан одномерный массив целых чисел. Напишите функцию, находящую число элементов, меньше заданного.

7. Задан одномерный массив вещественных чисел. Напишите функцию, находящую произведение минимального и максимального элементов.

8. Задан одномерный массив целых чисел. Напишите функцию, находящую число элементов, кратных заданному значению.

9. Задано два одномерных массива целых чисел. Напишите функцию, объединяющую их таким образом, что в результирующем массиве все элементы различны.

10. Задано два одномерных массива целых чисел. Напишите функцию, объединяющую их таким образом, что в результирующем массиве все элементы являются пересечением заданных.

Контрольные вопросы

1. Как реализовать бесконечный цикл с помощью оператора `While`?

2. Какая часть цикла `For` может быть пропущена?

3. Для чего нужна директива `Break`?

4. Что такое метка в JavaScript?

5. Опишите особенности использования локальных и глобальных переменных в пользовательских функциях.

6. Перечислите способы создания массива.

7. Что будет отображаться в браузере при выводе массива с пропущенными индексами?

8. Назовите отличия методов доступа к массиву от методов изменения. Приведите несколько примеров каждого метода.

Библиографический список

1. Дронов В.А. HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов. – СПб.: БХВ-Петербург, 2011. – 416 с.

2. Дунаев В.В. HTML, скрипты и стили. – 3-е изд. – СПб.: БХВ-Петербург, 2011. – 816 с.

3. Прохоренок Н., Дронов В.А. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера. – СПб.: БХВ-Петербург, 2010. – 912 с.

4. Интерактивный учебник по JavaScript. [Электронный ресурс]. URL: https://learn.javascript.ru/ (дата обращения: 01.03.2017).

Оглавление

Лабораторная работа № 2. Управляющие конструкции языка JavaScript. Стандартные объекты и функции ядра JavaScript... 1

1. Введение ... 1

2. Теоретическая часть ... 1

2.1. Управляющие конструкции языка JavaScript ... 1

2.2. Функции ... 6

2.3. Массивы с числовыми индексами ... 8

2.4. Методы изменения массива ... 10

2.5. Методы доступа ... 11

2.6. Дата и время ... 13

2.7. Объект Math ... 13

2.8. Стандартные функции верхнего уровня ... 14

3. Практическая часть ... 14

3.1. Самостоятельное задание ... 14

Контрольные вопросы ... 15

Библиографический список ... 15