Разработка клиентских сценариев с использованием JavaScript и библиотеки jQuery

STEP computer ACADEMY

Unit 2

Объекты. Массивы. Строки

Содержание

Объекты

Что такое объект?

Вы уже сталкивались с понятием объектов в ранее изученных языках программирования. Однако, лишний раз немного напомнить не повредит.

Объект — это некоторая конкретная реализация какой-то сущности. Например, у нас есть некоторая общая сущность яблоко. Объектом будет конкретное яблоко, лежащее перед нами. Или, например, сущность автомобиль, красный автомобиль марки AUDI, который мы видим перед собой в автосалоне, это объект. Ну и конечно сущности с точки зрения объектно-ориентированного программирования — это классы.

В JavaScript есть поддержка объектов и объектно-ориентированного программирования. Мы обязательно поговорим ещё об этом, но начнём мы немного с другой стороны. JavaScript даёт нам возможность посмотреть на объекты с двух сторон: первая — это объектно-ориентированный стиль, вторая — объекты, как ассоциативные массивы.

И начнем мы со второго подхода. Понятие ассоциативного массива вам уже встречалось ранее. Напомним, это массив, который состоит из пар «ключ-значение». Например, в качестве ключа может быть название стран, а в качестве значения названия столиц этих стран. Главное ограничение на такой массив состоит в том, что ключи должны быть уникальными. Посмотрим, как данная концепция реализована в JavaScript.

Первый шаг для нас — создание объекта. Делается это следующим образом:

```
// создаем пустой объект
// мы используем ключевое слово new для создания объекта
var obj = new Object();
// второй вариант создания объекта
var obj2 = {};
```

Вы можете использовать тот или иной механизм в зависимости от своих предпочтений. Для добавления пар «ключ-значение» в объект можно использовать два подхода. Первый подход через стандартный синтаксис массива. В примере ниже мы создаём объекта студента и заполняем его свойствами.

```
// создаем пустой объект
var student=new Object();
// Добавляем свойство в объект используя обычный
// синтаксис массивов
// ["ключ"]
student["Name"] = "Vasya";
// вместо двойных кавычек можно использовать одинарные
student['Age'] = 23;
alert(student["Name"]);
alert(student['Age']);
```

При втором способе вы добавляете свойства в объект, как будто вы уже ранее указывали их.

```
// создаем пустой объект
var firm={};
// создаём свойство внутри объекта
firm.Name = "Star Inc";
```

```
firm.Address = 'Somewhere street 5';
alert(firm.Name);
alert(firm.Address);
```

Вы можете использовать любой из двух способов, но если у вас возникнет необходимость хранить в качестве ключа строку, содержащую с пробелами, то тогда второй синтаксис не подойдет.

```
// создаем пустой объект
var dog = {};
// ниже правильное использование синтаксиса
dog['Name of dog'] = 'Caesar';
alert(dog['Name of dog']);
// строка ниже это синтаксическая ошибка
// dog.Name Of dog = 'Pit';
```

Удаление свойств

Добавленные к объекту свойства можно удалить. Для этого используется delete. После удаления свойства, ключ и значение безвозвратно исчезнут.

```
// создаем пустой объект
var cat = {};
// задаём значение свойствам
// синтаксис через точку
cat.Name = "Vasiliy";
// синтаксис, как при работе с массивами
cat["Age"] = 2;

// отображаем значения
alert(cat.Name);
```

```
// можем обращаться в любом синтаксисе
alert(cat.Age);
alert(cat["Age"]);

// удаляем свойства
delete cat.Name;
delete cat["Age"];

// при попытке показа значения мы увидим значение
// undefined
alert(cat.Name);
alert(cat.Age);
```

Проверка существования свойства внутри объекта

Вы уже знаете, что при отображении несуществующего свойства отображается значение undefined. А как проверить существует ли свойство в коде?

Для этого можно использовать конструкцию in. Она возвращает true, если свойство есть внутри объекта.

```
// создали объект
var obj={};
obj.Name = "Oleg";

// проверяем есть ли свойство Age
// in вернет false, так как этого свойства нет
if("Age" in obj){
    alert("Exists");
}
else{
    alert("Not exists");
}
```

Ещё один способ создания объекта со свойствами

Кроме уже известных вам механизмов создания объектов есть ещё один путь. Вы же не думали, что их всего два? Третий вариант создания объекта позволяет сразу создать большое количество свойств внутри объекта. Покажем на примере:

```
// создали объект студента
var student = {
    name: "Daria",
      lastName: "Kislicina",
age:23
};

/*
  То же самое, что и выше
  var student = {};
  student.name = "Daria";
  student.lastName = "Kislicina";
  student.age = 23;
*/
alert(student.age);
```

Этот путь очень удобен, когда вам нужно создать описать объект сразу. При этом у вас есть возможность добавить свойство позже, когда оно вам понадобится. При создании объекта можно внутрь вставить другой объект.

```
// создали объект студента
var student = {
    name: "Daria",
    lastName: "Kislicina",
    age:23,
    address:{
```

```
        street:"Tiraspolskaya 5",
        city:"Odessa",
        country:"Ukraine"
    }
};

alert(student.lastName);
alert(student.address.street);
alert(student.address.city);
```

Просмотр всех свойств внутри объекта

У вас существует возможность узнать все свойства объекта. Для этого необходимо использовать цикл с использованием конструкции in.

```
var rect={
    x:0,
    y:0,
    endX:10,
    endY:10
};

// в tempProperty будет попадать название свойства
// такое как x,y,endX,endY

for(var tempProperty in  rect){
    // отображаем название свойства
    alert(tempProperty);
    // значение свойства
    alert(rect[tempProperty]);
}
```

Массивы

Что такое массивы?

Мы надеемся, что когда вы читаете данные строки этот вопрос не стоит для вас остро, так как вы уже знакомы с этим понятием из других языков программирования, изученных ранее. Однако, все равно сделаем шаг и назад и вспомним, что такое массив.

Массив — это структура данных, которая группирует набор некоторых значений под одним именем. Для доступа к конкретному значению используется индекс. Индексация в массивах в JavaScript начинается с нуля. Сразу обращаем ваше внимание, что в JavaScript в массиве могут храниться значения разных типов.

В JavaScript массив может быть создан двумя способами. Начнем с конструкции Array.

Объект Array

Первый способ создания массива состоит в использовании конструкции new Array.

Ниже приводим возможные формы создания:

```
// создаем пустой массив
var arrayName = new Array();

// создаем массив заданной длины
var arrayName= new Array(Number length);

// создаем массив и сразу инициализируем его значениями
var arrayName = new Array(element1, element2,...
                          elementN);
```

Все три конструкции достаточно просты и не несут никаких сложностей для программиста. Рассмотрим пример записи значений в массив для каждой из форм.

```
// создали пустой массив
var arr = new Array();
// заполнили его значениями
arr[0] = 34;
arr[1] = 99;
arr[2] = 100;

// создали массив с длиной 3
var arr2 = new Array(3);

// заполнили его значениями
arr2[0] = 111;
arr2[1] = 56;
arr2[2] = 73;

// теперь длина массива увеличилась на 2 элемента
arr2[3] = 333;
arr2[4] = 999;

// создали массив и сразу записали в него три значения
var arr3 = new Array("music","guitar","apple");

// добавили ещё один элемент
arr3[3] = "lemon";
```

Обратите внимание массивы в JavaScript автоматически увеличиваются в размере, когда это необходимо. Это значит, что вам не придется задумываться о динамическом выделении памяти.

Для того, чтобы узнать длину массива нужно воспользоваться встроенным свойством массива под названием length.

```
var arr = new Array(10,20);

// отображаем значение нулевого элемента
// 10
alert(arr[0]);

// с помощью alert можно показывать весь массив сразу
// элементы массива будут перечислены через запятую
alert(arr);

// показываем длину массива
// 2
alert(arr.length);
```

При использовании свойства length нужно помнить об одном важном моменте. Свойство length — это не количество элементов массива, а значение последнего индекса +1.

```
var arr = new Array();

// реальных заполненных элементов один
arr[499] = 86;

// на экране 500
// length это последний индекс + 1
alert(arr.length);

// при обращении к неинициализированному элементу
// отобразится undefined
alert(arr[0]);
```

Вы уже знаете, что массивы растут в размере самостоятельно, когда это требуется кодом скрипта. А что делать в том случае, если вам нужно уменьшить размер масси-

ва? Ответ очевиден! Надо уменьшить значение length до нужной длины.

```
var arr = new Array(11, 74, 35);

// делаем размер массива равным 2
// значение 35 потеряно навсегда
arr.length = 2;

// на экране 11,74
alert(arr);

// а теперь размер массива 5, но
// заданы значения только двум элементам
// arr[0] = 11
// arr[1] = 74
arr.length = 5;
alert(arr);

// теперь размер массива 0
// все значения утеряні навсегда
arr.length = 0;
```

Создание массива второй способ

Первый способ создания массивов немного громоздкий. Второй способ привычней для нас. Рассмотрим его:

```
// создаём пустой массив
var arrayName = [];

// создаём массив с набором значений
var arrayName = [element1, element2,..., elementN];
```

Интересным моментом является использование [] для создания массива. Раньше вы сталкивались с исполь-

зованием {} для той же цели. Теперь попробуем на примере новые конструкции.

```
// создаём пустой массив
var arr = [];

// записали в него 2 элемента
arr[0] = 11;
arr[1] = 12;
// 11,12
alert(arr);

// создаём массив с тремя элементами
var arr= [88,99,111];
// 88,99,11
alert(arr);

// создаём массив с тремя элементами
var cars = ["BMW","Audi","Toyota"];
// "BMW","Audi","Toyota"
alert(cars);
// 3
alert(cars.length);
```

По этому коду можно сделать простой вывод: такая форма создания массива удобней первого способа с Array.

Обращение к элементам массива

Мы уже умеем обращаться к элементам массива используя индекс. Добавим в наш багаж пример перебора массива с помощью цикла.

```
var arr = [2,9,33,1];
var amt = 0;
// считаем в цикле сумму элементов массива
```

```
for(var i = 0;i<arr.length;i++){
    amt+=arr[i];
}

// результат 45
alert(amt);
```

И ещё пример. Отобразим содержимое массива с элементами разного типа.

```
var arr = [33,"sun",12,"planet"];
for(var i = 0;i<arr.length;i++){
    alert(arr[i]);
}
```

Теперь настало время создать двумерный массив.

```
// создали двумерный массив 2 строки 3 столбца
// 1 3 5
// 2 7 8
var arr = [
    [1,3,5],
    [2,7,8]
];

// 1
alert(arr[0][0]);
// 8
alert(arr[1][2]);
```

Обычные переменные в JavaScript передаются внутрь функции по значению. Это значит, что при передаче переменной в функцию создаётся копия переменной, если вы измените значение этой копии это никак не повлияют

на оригинальную переменную. Массивы же передаются в функцию по ссылке. Так происходит, потому что массив — это объект, а объекты всегда передаются внутрь функций по ссылке. Отсюда можно сделать вывод, что изменения значений элементов массива сохраняются при выходе из функции.

```
// функция записывает новое значение по указанному
// индексу
function SetValue(arr,index,newValue){
    arr[index] = newValue;
}
var arr = [88,11,3];

// 88,11,3
alert(arr);
SetValue(arr,0,999);

// 999,11,3
alert(arr);
```

Свойства и методы массивов

У массивов в JavaScript есть большое количество встроенных методов. Мы познакомимся с некоторыми из них.

Начнем с методов для поиска. Это indexOf и lastIndexOf. Первый ищет совпадение в массиве слева направо. Сигнатура метода:

```
name_of_array.indexOf(what_to_search[, fromIndex])
```

- what_to_search — значение, которое мы ищем в массиве;

- fromIndex — необязательный параметр, который используется для указания стартового индекса. Если мы не указываем его, то поиск начнется с нулевого индекса.

Если искомое значение найдено метод возвращает индекс найденного значения, если же значения нет в массиве, то метод вернет -1.

Продемонстрируем работу этого метода на примерах:

```
var arr = [1,45,-3,78,1];

// ищем значение 45
var index = arr.indexOf(45);

// на экране индекс 1
alert(index);

// ищем значение, которого нет в массиве
index = arr.indexOf(99);

// на экране -1, так как 99 нет в массиве
alert(index);
```

А теперь давайте посмотрим пример подсчета сколько раз некоторое искомое значение встречается в массиве.

```
var arr = [12,45,-3,82,12,78,12];

// счетчик для подсчета количества раз вхождения
// искомого значения в массив
// искать будем значение 12
var counter = 0;
var index =  arr.indexOf(12);
while(index != -1){
    counter++;
```

```
    // двигаемся дальше по массиву за счет изменения
    // индекса на значение индекс+1
    index = arr.indexOf(12,index+1);
}
// на экране 3
alert(counter);
```

Метод lastIndexOf работает по схожему индексу, но ищет справа налево. Это значит, что поиск начинается с последнего доступного индекса. Сигнатура метода:

```
name_of_array.indexOf(what_to_search[, fromIndex])
```

- what_to_search — значение, которое мы ищем в массиве;
- fromIndex — необязательный параметр, который используется для указания стартового индекса. Если мы не указываем его, то поиск начнется с самого последнего индекса.

Если искомое значение найдено метод возвращает индекс найденного значения, если же значения нет в массиве, то метод вернет -1.

Давайте на примере разберем отличия этого метода.

```
var arr = [12,45,-3,82,12,78,12];
// ищем 12 справа налево
var index =  arr.lastIndexOf(12);
// на экране 6
alert(index);
index =  arr.lastIndexOf(77);
// на экране -1
alert(index);
```

А теперь посчитаем сколько раз встречается искомое значение в массиве при помощи lastIndexOf.

```
var arr = [12,45,-3,82,12,78,12];
// счетчик для подсчета количества раз вхождения
// искомого значения в массив
// искать будем значение 12
var counter = 0;
var index =  arr.lastIndexOf(12);
while(index != -1){
    counter++;
    // мы проверяем на ноль
    // так как ниже начинаем с index-1
    // при 0 мы получим старт -1
    // для метода lastIndexOf отрицательный индекс
    // означает искать с конца массива
    if(index == 0)
        break;
    // двигаемся дальше по массиву за счет изменения
    // индекса на значение индекс-1
    index = arr.lastIndexOf(12,index-1);
}
// на экране 3
alert(counter);
```

При работе с данными очень важно иметь механизм их сортировки. И такой механизм представлен в виде метода sort, который имеет следующую сигнатуру:

```
name_of_array.sort([compareFunc])
```

- what_to_search — значение, которое мы ищем в массиве;
- compareFunc — необязательный параметр, имя пользовательской функции, которая будет использована

для сортировки массива. Если её не указывать данные в массиве будут отсортированы по правилам сортировки строк

Начнем в наших примерах с сортировки по умолчанию.

```
var arr = [10,1,3,33,6];
arr.sort();
// 1 10 3 33 6
alert(arr);
```

В результате сортировки по умолчанию 10 стоит перед 33, так получилось из-за строковой сортировки. Если вам нужна числовая сортировка, то нужно будет реализовать функцию сортировки. Возможно вы уже сталкивались с таким решением в других языках программирования. Общая форма данной функции:

```
function name_of_function(name_of_var1, name_of_var2){
    body_of_function
}
```

Вы можете назвать функцию любым именем. Функция должна принимать два параметра. Это две значения массива, которые в данный момент сравниваются алгоритмом сортировки. Предположим вы хотите сортировать элементы массива по возрастанию, тогда правила сортировки для вас такие,

- если name_of_var1 > name_of_var2, возвращаемое значение из функции положительное (обычно используют 1);

- если name_of_var1 < name_of_var2, возвращаемое значение из функции отрицательное (обычно используют -1);
- если равны, то обычно возвращают 0.

Для сортировки по убыванию в первом случае возвращайте отрицательное значение, а во втором положительное.

Рассмотрим пример сортировки данных по возрастанию:

```
function compareFunc(a,b){
    if(a>b)
        return 1;
    else if(b>a)
        return -1;
    else
        return 0;
}
var arr = [10,1,3,33,6];

// 1 3 6 10 33
arr.sort(compareFunc);
```

А теперь сортировка по убыванию

```
function compareFunc(a,b){
   if(a>b)
     return -1;
   else if(b>a)
     return 1;
   else
    return 0;
}
var arr = [10,1,3,33,6];
```

```
// 33 10 6 3 1
arr.sort(compareFunc);
alert(arr);
```

Рассмотрим следующую задачу: у нас есть строка и нам нужно её конвертировать в массив, разбив на элементы массива на основе какого-то разделителя. Для того, чтобы это сделать используем метод строки под названием split. Его сигнатура:

```
name_of_string.split(separator)
```

- separator — сепаратор с его помощью мы разбиваем строку на элементы массива. В результате работы метода возвращается массив результата.

Этот метод можно использовать, например, при работе с текстом. Разобьем строку с помощью вызова split.

```
var str = "apple,onion,strawberry";
// разбиваем на основании ,
// в массиве будет три элемента apple onion strawberry
var arr = str.split(',');
// apple на экране
alert(arr[0]);
```

Обратите внимание, что split это строковый метод. У массива есть обратный метод join. Он используется для конвертирования массива в строку. Например:

```
var arr = ["bmw","audi","opel"];
// создаем строку
// в качестве разделительного символа между элементами
```

```
// массива указываем *
var str = arr.join("*");
// bmw*audi*opel
alert(str);
// если не указать разделитель, то будет использована,
var str2 = arr.join();
// bmw,audi,opel
alert(str2);
```

Мы привели лишь часть методов массива в JavaScript. С остальными вы можете ознакомиться тщательно изучив MSDN или документацию по ссылке.

Строки

Объект String

В JavaScript не существует отдельных типов для строк и символов. Есть только строковый тип и если вам нужно работать с одним символом вы создаете строку с одним символом. Содержимое строки может быть заключено в двойные или одинарные кавычки. Для JavaScript между ними нет разницы. Вы можете выбрать любой стиль, хотя с нашей точки зрения использование двойных кавычек более привычно. Пример создания строк:

```
// используем двойные кавычки
var str = "Test string";
alert(str);

// используем одинарные кавычки
var str2 = 'New string';
alert(str2);
```

Внутренним форматом хранения строк в JavaScript является Unicode. И при этом совершенно неважно какая у вас кодировка вашей страницы. Вы можете включать в свою строку специальные символы. Например, escape-последовательности. Пример строк с такими символами:

```
// \t – табуляция
var str = "Sun\t is going \\down\\";
// Sun is going \down\
alert(str);
```

```
// для вставки кавычек в строку надо использовать \"
var str2 = "\"Yes\"";

// на экране "Yes"
alert(str2);
```

Для доступа к конкретному элементу строки необходимо использовать уже известные вам [].

```
var str = "trees and fruits";

// на экране trees and fruits
alert(str);

// на экране t
alert(str[0]);

// на экране s
alert(str[4]);
```

Строки в JavaScript являются неизменяемыми объектами. Это значит, что после создания строки вы не можете изменить i-ый элемент строки (например, элемент по индексу 0 или 5). Вы можете пересоздать строку только целиком!

```
var str = "trees and fruits";

// на экране trees and fruits
alert(str);
str[0] = "Z";

// на экране все равно t
alert(str[0]);
```

```
str[4] = 'W'

// на экране s
alert(str[4]);
str = "Here is a car";

// на экране Here is a car
alert(str);
```

Для конкатенации строк используется оператор +.

```
var first = "boat";
var second = "river";
var result = first + " and " + second;

// boat and river
alert(result);
```

Свойства и методы String

У строки есть большое количество свойств и методов, которые очень полезны для решения той или иной задачи. Начнем с самого очевидного: свойство length используется для получения длины строки:

```
var str = "gold";

// 4
alert(str.length);
```

Вы не можете уменьшить или увеличить длину строки задав значение length явно. Это не приведет к желаемому результату.

Метод charAt является аналогом [ ] и используется для доступа к i-му элементу массива. Отличие между charAt и [ ] состоит в том, что при обращении к несуществующему индексу charAt возвращает пустую строку, а [ ] значение undefined.

```
var str = "gold";

// g
alert(str.charAt(0));

// пустая строка
alert(str.chartAt(15));

// undefined
alert(str[15]);
```

Методы toLowerCase и toUpperCase изменяют регистр букв. Метод toLowerCase меняет регистр на нижний, а toUpperCase меняет регистр на верхний. Обратите внимание, что новое значение возвращается из методов, но при этом содержимое самой строки не меняется.

```
var str = "FootballL";
var newStr = str.toLowerCase();

// football
alert(newStr);

// FootballL
alert(str);
```

Ранее из урока вы уже узнали о методах indexOf и lastIndexOf, которые использовались для поиска по

массиву. У строки есть свои версии этих методов, которые ведут себя также, как и их аналоги в массиве. Пример поиска значения в строке:

```
var str = "earth and sun";
var index = str.indexOf("sun");

// значение индекса равно 10
alert(index);
index = str.indexOf("moon");

// значение индекса равно -1
// так как moon нет в строке
alert(index);
```

А теперь посчитаем сколько раз некоторое слово встречается в строке:

```
var str = "test it is test sun test no";
var counter = 0;
var wordToFind = "test";
var index = str.indexOf(wordToFind);
while(index != -1){
    counter++;
      index = str.indexOf(wordToFind,index+1);
}
// 3
alert(counter);
```

Схожий пример поиска вы уже видели в разделе о массивах.

Методы substr, substring используются для получения подстроки из строки.

Начнем с substring(start, [end]). Метод возвращает подстроку начиная с индекса start, но не включая ин-

декс end. Если end не указан, то возвращаем подстроку до конца строки.

```
var str = "Some value";

// end — необязательный параметр, который можно опустить
var newStr = str.substring(2);

// me value
alert(newStr);
newStr = str.substring(1,3);

// om
alert(newStr);
```

Метод substr(start, [length]) работает немного по другому. Он возвращает подстроку начиная с start, при этом можно указать длину подстроки во втором параметре. Если же длина не указана, то возвращается подстрока до конца оригинальной строки.

```
var str = "Some value";

// length — необязательный параметр, который можно
// опустить
var newStr = str.substr(2);

// me value
alert(newStr);
newStr = str.substr(1,3);

// ome
alert(newStr);
```

Для сравнения строк с учетом локали используется метод localCompare(compareValue). Если строки между

собой равны метод вернет 0, если строка, которая вызывала метод больше строки в параметре метод вернет 1, иначе -1. Принципы сравнения такие же, как у известной вам из языка C функции strcmp.

```
var str = "cheese";

// попадем в if, так как строки равны между собой
if(str.localeCompare("cheese")==0){
    alert("Strings are equal! Cheese!");
}
else{
    alert("Strings are not equal! Not cheese!");
}

str = "Fb";

// первая строка больше, так как сравнение происходит
// посимвольно до первого несовпадения
// F больше по числовому коду чем f
// поэтому первая строка больше
if(str.localeCompare("fb")>0){
    alert("First is greater");
}
else{
    alert("Equal or less than second string");
}
```

Задержки и интервалы

Периодический вызов функций

Иногда возникает необходимость вызвать функцию через определенное количество время. Например, если вы реализуете онлайновую игру и дали игроку 60 секунд на решение пазла, вам нужно иметь механизм, который отсчитает эти 60 секунд, после чего игра сообщит пользователю, что время вышло. В обычной жизни такой механизм называется таймер. Вы можете ежедневно наблюдать работу таймера в микроволновой печи, когда производите разогрев чего-либо в ней.

Для реализации таймера в JavaScript существует набор функций. Начнем с функции установки таймера.

```
setTimeout(функция/код, задержка, аргумент1, аргумент2, …)
```

Функция setTimeout используется для установки таймера. Функция/код — функция или код, который запустится через указанный промежуток времени.

Задержка — время задержки для таймера. После истечения времени будет запущена функция/код из первого аргумента. Задержка указывается в миллисекундах. Для справки: 1000 миллисекунд = 1 секунде.

Аргумент1, аргумент2, … — аргументы, которые можно передать в функцию, которая сработает, когда указанная задержка закончится.

Функция setTimeout возвращает идентификатор установленного таймера. Его можно использовать для того, чтобы отменить срабатывание таймера.

Начнем знакомство с таймером с самых простых примеров. В этом примере кода таймер сработает через секунду и будет вызвана функция HelloWorld, указанная в параметре.

```
function HelloWorld(){
    alert("Hello world!");
}
setTimeout(HelloWorld, 1000);
```

Попробуем передать аргументы в функцию:

```
function Sum(a,b){
   alert(a+b);
}
setTimeout(Sum, 1000,1,2);
```

Мы передали значения 1 (попало в параметр a) и 2 (попало в параметр b). Функция посчитала сумму двух чисел и показала её в окне сообщения.

Когда мы описывали параметры для setTimeout мы говорили, что первый параметр может быть функцией или просто кодом. Например:

```
setTimeout("alert('hi')", 1000);
```

Здесь в качестве кода используется alert. Указывать код в таком виде не рекомендуется, так как это нечитабельно и минимизаторы кода при обработке такого фрагмента могут неправильно его интерпретировать. Альтернативой является использование безымянных (анонимных) функций.

```
    setTimeout(function(){
    alert("Hi!");
    }, 1000);
```

В примере выше мы создали анонимную функцию с одной строкой кода для отображения окна сообщения. Такие анонимные функции можно создавать и с параметрами.

```
setTimeout(
    function(a,b){
        alert(a*b);
    },
    1000, 3, 7
);
```

После установки таймера возвращается идентификатор. Его можно использовать для отмены вызова функции.

```
var id = setTimeout(function(){
    alert("Boom!")},3000
);
alert("id timer: "+id);
```

В этом примере мы отобразили идентификатор таймера. В следующем примере мы используем полученный идентификатор для отмены таймера. Функция отмены таймера clearTimeout.

```
clearTimeout(идентификатор_таймера)
 var id = setTimeout(function(){
    alert("Boom!")},50000
);
clearTimeout(id);
```

Мы установили таймер на 50 секунд, а потом отменили его. Естественно в результате наших действий работа таймера будет остановлена.

А что же делать, если нам необходимо срабатывание некоторой функции через определенные интервалы времени? Механизм, который мы разобрали выше позволяет создать функцию, которая будет вызвана единожды. Для решения нашей проблемы можно использовать два пути.

Первый способ это использование функции setInterval.

```
setInterval(функция/код, интервал_времени, аргумент1,
            аргумент2, …)
```

Посмотрев на функцию можно легко сделать вывод о том, что она полностью схожа с setTimeout. Отличие только в принципе действия. В отличии от setTimeout функция/код, указанная в первом параметре будет регулярно вызываться через указанный интервал времени. Для остановки вызова функции нужно будет вызвать функцию clearInterval. Попробуем использовать интервальный механизм.

```
setInterval(
    function(){
        alert("Boom!");
    },
    2000);
```

Функция, указанная в первом параметре будет вызываться каждые две секунды. В этом коде мы не вызвали clearInterval поэтому окно сообщения с надписью Boom будет вызываться каждые две секунды, пока окно или

вкладка браузера не будет закрыта. Теперь добавим вызов clearInterval.

```
var id = setInterval(IntervalFunc,2000);
var counter = 0;

function IntervalFunc(){
    if(counter == 3){
        clearInterval(id);
        return;
    }
    counter++;
    alert("Boom");
}
```

Окно сообщения будет показано три раза, после чего мы остановим интервальную функцию. Альтернативой использованию setInterval является рекурсивный вызов setTimer. Например:

```
var id = setTimeout(TimeOutFunc, 2000);
var counter = 0;

function TimeOutFunc() {
    // если таймер сработал уже трижды
    // останавливаем процесс
    if (counter == 3) {
        clearTimeout(id);
        return;
    }
    counter++;
    alert("Boom Timer");

    // ставим занова таймер на две секунды
    id = setTimeout(TimeOutFunc, 2000);
}
```

Принцип использования таймера вместо интервалов прост: внутри таймерной функции мы заново ставим таймер. В нашем примере мы заново устанавливали таймер на две секунды внутри таймерной функции. Когда наш цикл работы исполнится три раза мы вызываем clearTimeout и выходим из таймерной функции. При установке нового таймера мы не обязаны указывать ту же задержку, что и при первом вызове. Например:

```
var id = setTimeout(TimeOutFunc, 2000);
var counter = 1;
function TimeOutFunc() {
    alert("Boom Timer");
    switch(counter){
        case 1:
              id = setTimeout(TimeOutFunc, 5000);
          break;
         case 2:
              id = setTimeout(TimeOutFunc, 10000);
          break;
         case 3:
              clearTimeout(id);
           return;
    }
     counter++;
}
```

В нашем случае у каждого из таймеров своя задержка: 2, 5, 10 секунд.

Для создания интервальной функции можно использовать setInterval или setTimeout. Выбор за вами. Мы можем добавить лишь то, что setTimeout несколько гибче, так как у вас есть возможность каждый раз указывать новую длительность задержки.

Использование математических возможностей

Объект Math

При решении задач из области программирования вам могут понадобиться математические возможности, встроенные в JavaScript. Как же можно воспользоваться ими? Сложно ли это сделать? Ответ: нет. Для того, чтобы использовать математические возможности JavaScript вам необходимо обратиться к встроенному объекту Math. Внутри этого объекта вы найдете целую мириаду функций для решения той или иной математической проблемы, стоящей перед вами. Давайте начнем знакомство с внутренним устройством этого объекта.

Свойства и методы объекта Math

Мы не будем рассматривать все свойства и методы объекта Math. Начнем с некоторых свойств:

- Math.PI — возвращает число ПИ (приблизительно 3.14);
- Math.E — возвращает число Эйлера (приблизительно 2.718);
- Math.SQRT2 — возвращает квадратный корень из двух (приблизительно 1.414);
- Math.SQRT1_2 — возвращает квадратный корень из одной второй (приблизительно 0.707).

Пример использования:

```
alert(Math.PI);
alert(Math.E);
```

Теперь давайте рассмотрим некоторые методы:

- Math.ceil(x) — возвращает округление параметра x вверх до ближайшего целого;
- Math.floor(x) — возвращает округление параметра x вниз до ближайшего целого;
- Math.round(x) — возвращает округленное значение параметра x до ближайшего целого (если дробная часть больше или равна 0,5 тогда округление вверх, иначе вниз);
- Math.pow(x, y) — возвращает значение x возведенное в степень y;
- Math.sqrt(x) — возвращает квадратный корень из x;
- Math.min() и Math.max — возвращают соответственно минимум и максимум из переданных параметров;
- Math.abs(x) — возвращает модуль переданного параметра x;

Пример использования функций по округлению и усечению значений:

```
var res = Math.ceil(4.3);
// 5
alert(res);
res = Math.round(4.3);
// 4
alert(res);
```

```
res = Math.round(4.5);
// 5
alert(res);
res = Math.floor(4.5);
// 4
alert(res);
```

Пример использования функций по поиску максимума и минимума:

```
var res = Math.min(1,4,-1,-9,20);
// -9
alert(res);
res = Math.max(21,3,4,5,6,7);
// 21
alert(res);
```

Мы уверены, что знакомство и использование других математических методов не составит для вас сложности.

Случайные числа

В жизни любого программиста наступает момент, когда приходится иметь дело со случайными числами. Например, вам необходимо создать программное обеспечение для лотереи или для игровых автоматов. В этом случае вам нужен будет механизм, который будет генерировать вам случайные числа. Для генерации псевдослучайных чисел в JavaScript используется метод random объекта Math.

- Math.random() — возвращает псевдослучайное число в диапазоне от 0 до 1;

Пример использования:

```
var res = Math.random();
// псевдослучайное число
alert(res);
res = Math.random();
// псевдослучайное число
alert(res);
```

Обратите внимание, что число псевдослучайное. Это означает, что число, которое будет возвращено будет сгенерировано алгоритмически. Обычно в основе алгоритмов, генерирующих случайные числа, лежит понятие начальной точки. По умолчанию обычно при первом вызове функции для генерации случайного числа в качестве начальной точки берут количество миллисекунд, прошедших с 1 января 1970 года. В некоторых языках вы можете изменить начальную точку. В JavaScript данная возможность скрыта. Каким образом можно сгенерировать псевдослучайное число в другом диапазоне, отличном от 0 и 1. Для решения этой задачи используется связка random и floor. Например:

```
// случайное значение от 0 до 9
alert(Math.floor(Math.random()*10));
// случайное значение от 0 до 11
alert(Math.floor(Math.random()*11));
// случайное значение от 1 до 10
alert(Math.floor(Math.random()*10)+1);
```

Как видно из примера выше использование случайных чисел не несет большой сложности.

Unit 2.

Объекты. Массивы. Строки