Урок 3

Циклы, массивы, структуры данных.

Реализация циклов в Javascript. Введение в методы и свойства. Знакомство с массивами.

Что такое цикл

Цикл - разновидность управляющей конструкции в высокоуровневых языках программирования, предназначенная для организации многократного исполнения набора инструкций. Говоря простым языком, цикл – это конструкция, которая заставляет определённую группу действий повторять до

наступления нужного условия. К примеру:

- Опрашивать клиентов пока не наберётся 100 успешных ответов;
- Читать строчки из файла, пока не доберёмся до конца самого файла.

Набор этих инструкций называется телом цикла. Каждое выполнение тела цикла называется итерацией. Условие, которое определяет, делать ли следующую итерацию или завершить цикл, называется условием выхода. Очень удобно знать, сколько итераций уже сделано – т.е. хранить их число в переменной. Такая переменная называется счётчиком итераций цикла, но очевидно, что она не является обязательной для всех циклов.

Итак, цикл состоит из следующих шагов

1. Инициализация переменных цикла в начале цикла.
2. Проверка условия выхода на каждой итерации (до или после неё).
3. Исполнение тела цикла на каждой итерации.
4. Обновление счётчика итераций.

Кроме того, в большинстве языков программирования есть инструменты досрочного прерывания всего цикла или же выполнения текущей итерации.

Циклы в JavaScript

Как и у любого языка программирования, в JavaScript также есть циклы. Всего есть четыре цикла: while, do/while, for и for/in. Каждому из них посвящён один из следующих подразделов. Одно из обычных применений циклов - обход элементов массива.

Цикл while

Цикл while является примером цикла с предусловием. Это цикл, который выполняется, пока истинно некоторое условие, указанное перед его началом. Поскольку условие проверяется до выполнения самой первой итерации, вполне может не выполниться ни одной итерации, если условие изначально ложно.

```
while( Условие ) {
  Операторы;
}
```

Тело цикла, содержащее операторы, будет выполняться до тех пор, пока истинно условие, указанное в начале цикла. Алгоритм такого цикла в сравнении с базовым будет выглядеть так:

1. Проверка условия.
2. Выполнение тела, если условие истинно. Выход, если условие ложно.

Для управления циклом обычно требуется одна или несколько переменных. К примеру, некое значение boolean, которое обращается в false при достижении некоего граничного условия. Или же целочисленное значение, каждый раз увеличиваемое на единицу, пока не достигнет определённого значения. На практике:

```
var n = 10;
var i = 1;
while (i <= n) {
```

```
  console.log(i++);
}
```

Здесь мы использовали новую для нас конструкцию console.log(). Это удобная функция для вывода отладочной информации, не мешая пользователю всплывающими сообщениями. Посмотреть её можно в отладчике браузера на вкладке Console. Обратите внимание, что данная функция совместима с IE от 8 версии и выше.

В начале переменной ~~count~~ i присваивается значение 1 ~~0~~, а затем её значение увеличивается каждый раз, когда выполняется тело цикла. После того как цикл будет выполнен 10 раз, выражение вернёт false (т.е. переменная ~~count~~ 1 уже ~~не меньше~~ больше 10), инструкция while завершится, и интерпретатор перейдёт к следующей инструкции в программе. Большинство циклов имеют переменные-счётчики, аналогичные ~~count~~ i. Чаще всего в качестве счетчиков цикла выступают переменные с именами i, j и k, хотя для того, чтобы сделать программный код более понятным, следует давать счётчикам более наглядные имена.

Цикл do..while

Цикл do…while – это уже цикл с постусловием, работающий по алгоритму:

1. Выполнение блока операторов.
2. Проверка условия.
3. Выход, если условие ложно.

Цикл с постусловием – это цикл, в котором условие проверяется после выполнения тела цикла. Отсюда следует, что тело всегда выполняется хотя бы один раз.

В следующем примере единица будет выведена, даже если N=0:

```
var n = 10;
var i = 1;
do{
  console.log(i++)
} while (i <= n);
```

Цикл do...while используют достаточно редко ввиду его громоздкости и плохой читаемости. Также дело тут в том, что ситуация, когда необходимо гарантировать хотя бы одно выполнение тела цикла, экзотична.

Цикл for

Цикл for – это цикл со счётчиком. Сам по себе этот цикл представляет прекрасный пример лаконичной организации кода, имея максимально схожий вид в большинстве языков программирования.

Цикл for значительно упрощает объявление циклов. Как Вы уже увидели, даже без прямого на то требования, циклы имеют счётчик. Он объявляется перед началом цикла и проверяется на каждой итерации, изменяя своё значение внутри тела цикла согласно установленным программистом правилом. Инициализация, проверка и обновление - это три ключевых операции, выполняемых со счетчиком. Цикл for сводит эти три шага воедино:

```
for(инициализация; проверка; инкремент)
```

```
{
   инструкция
}
```

Если провести аналогию с известным нам уже циклом while, получится вот такой псевдокод:

```
инициализация;
while(проверка)
{
   инструкция;
   инкремент;
}
```

Итак, в цикле for перед началом цикла выполняется инициализация, в которой обычно и объявляется переменная-счётчик. Проверка вычисляется в начале каждой итерации цикла. Если она возвращает true, цикл продолжается, иначе – останавливается. По окончанию итерации производится инкрементирование счётчика.

И снова в цикле выведем числа от 1 до 10, но уже с применением цикла for

```
for (var count = 1; count <= 10; count++)
        console.log(count);
```

При этом любое определяющее выражение можно опустить. Например:

```
var count = 0;
for (;count <= 10;)
        console.log(count++);
```

Цикл for/in

Цикл for/in использует ключевое слово for, но он в корне отличается от обычного цикла for. Цикл for/in имеет следующий синтаксис:

```
for (переменная in объект)
{
   инструкция
}
```

В качестве переменной обычно используется имя переменной, но точно так же можно использовать инструкцию var, объявляющую единственную переменную. Параметр объект - это выражение, возвращающее объект. И как обычно, инструкция - это инструкция или блок инструкций, образующих тело цикла.

Пока мы не знакомы с объектами, но цикл for/in позволяет обходить именно их

```
var obj = {name:'Alex', password:'12345' };
for (var i in obj)
{
        // Вывести значение каждого свойства объекта
        console.log(obj[i]);
}
```

Бесконечный цикл и выход из шагов цикла

Давайте вспомним нашу игру с прошлого занятия. Для решения постоянного запрашивания данных у пользователя нам приходилось применять рекурсию, что на самом деле неправильно, т.к. мы начинали не оптимально расходовать память. А вот применив т.н. бесконечный цикл, можно было бы решить эту проблему изящнее и проще.

Бесконечный цикл – это настоящий бич неопытных программистов. Если в него войти без соответствующего контроля, то с каждой итерацией скорее всего будет потреблять больше и больше памяти, и в конечном итоге он «повесит» браузер пользователя. Но в умелых руках бесконечный цикл можно обернуть на пользу.

Бесконечным считается цикл вида:

```
while(true){ … }
```

Как же ими управлять, если условие выхода никогда не вернёт false? Для выхода из всего процесса цикла используется оператор break.

```
var n = 10;
var i = 1;
while (true){
  console.log(i++);
  if (i > n)
    break;
}
```

В данном примере мы выводим числа от 1 до 10, но при помощи бесконечного цикла. Каждый раз в конце цикла проверяется состояние переменной i. Как только она станет равна 11, вызовется оператор break, который мгновенно завершает выполнение цикла и приступает к выполнению инструкций, идущих после объявления цикла в коде.
Не всегда нужно останавливать весь цикл. Иногда нужно пропустить итерацию и вернуться к проверке условия, но уже перед следующим выполнением тела цикла. Для этого используется оператор continue.

```
var n = 10;
var i = 1;
```

```
while (true){
  if (i % 2 == 0)
    continue;
  console.log(i++);
}
```

Так или иначе, правильнее считается не использовать операторы прерывания цикла, а возлагать логику управления на условие. Старайтесь организовывать циклы именно так.

Массивы

Массив – это именованный набор однотипных переменных. Определение не очень понятно, поэтому гораздо проще обрисовать массив в виде большого шкафа со множеством ящиков. Сам шкаф – это массив, а ящики в этом шкафу – это элементы массива.

В классическом виде нумерация элементов (ящиков) начинается с нуля. Ключи не могут повторяться, они уникальны. Таким образом, массив может быть определён как переменная, которая может хранить внутри себя не одно, а множество значений.

Отсчёт индексов массивов начинается с нуля и для них используются 32-битные целые. Массивы в JavaScript являются динамическими: они могут увеличиваться и уменьшаться в размерах по мере необходимости; нет необходимости объявлять фиксированные размеры массивов при их создании или повторно распределять память при изменении их размеров.

Самый простой способ создать массив в JS – использовать литерал. Он представляет собой простой список разделённых запятыми элементов массива в квадратных скобках. Значения в литерале массива не обязательно должны быть константами - это могут быть любые выражения, в том числе и литералы объектов:

```
var empty = [];                                 // Пустой массив
var numbers = [2, 3, 5, 7, 11];                 // Массив с пятью числовыми элементами
var misc = [ 1.1, true, "a", ];                 // 3 элемента разных типов + завершающая запятая
var base = 1024;
var table = [base, base+1, base+2, base+3];     // Массив с переменными
var arrObj = [[1,{x:1, y:2}], [2, {x:3, y:4}]]; // 2 массива внутри, содержащие объекты
```

Другой способ создания массива состоит в вызове конструктора Array(). Вызвать конструктор можно тремя разными способами:

```
var arr = new Array();
var arr = new Array(10);
ar arr = new Array(5, 4, 3, 2, 1, "тест");
```

Но мало создать массив. Нужно уметь читать его элементы, изменять и удалять их.

```
// Создать массив с одним элементом
var arr = ["world"];
// Прочитать элемент 0
var value = arr[0];
// Записать значение в элемент 1
arr[1] = 3.14;
// Записать значение в элемент 2
i = 2; arr[i] = 3;
// Записать значение в элемент 3
arr[i + 1] = 'привет';
// Прочитать элементы 0 и 2, записать значение в элемент 3
arr[arr[i]] = arr[0];
```

У каждого массива есть автоматически подсчитываемое свойство размера – length. Это может пригодиться при обходе этого массива в цикле, что мы сделаем немного позже.

Наиболее простой способ добавления элементов в массив состоит в том, чтобы присваивать значения по индексам. Но для добавления одного или нескольких элементов в конец массива можно также использовать метод push()

```
var arr = [];              // Создать пустой массив
arr.push('zero');          // Добавить значение в конец
arr.push('one',2);         // Добавить еще два значения
```

Для вставки элемента в начало массива можно использовать метод unshift(). Но помните, что при этом уже существующие элементы изменяют свои индексы и смещаются в позиции с более высокими индексами.

Удалять элементы массива можно с помощью оператора delete, как обычные свойства объектов:

```
var arr = [1,2,'three'];
delete arr[2];
```

При удалении элемента не происходит смещения элементов, а удалённому индексу присваивается соответствующее значение ~~undefined~~ empty.

Массивы имеют метод pop() (противоположный методу push()), который уменьшает длину массива на 1 и возвращает значение удаленного элемента. Также имеется метод shift() (противоположный методу unshift()), который удаляет элемент в начале массива. В отличие от оператора delete, метод shift() сдвигает все элементы вниз на позицию ниже их текущих индексов.

Если мы хотим обработать каждый элемент массива, то мы совершаем так называемый обход массива в цикле. Самый простой способ обойти массив – использовать цикл for.

```
var arr = [];
```

```
arr.push('zero');
arr.push('one',2);
for (var i = 0; i < arr.length; i++) {
   console.log(arr.length);
}
```

Очевидно, для многоуровневых массивов можно применять вложенные циклы, если на момент старта точно известен уровень вложенности. Иначе нужно применять рекурсивные функции.

Практикум

Быки и коровы

Усложним нашу предыдущую игру. Ваш соперник (компьютер, например), загадывает 4-значное число, состоящее из неповторяющихся цифр. Ваша задача - угадать его. Кстати, число ходов можно и ограничить. В качестве подсказок выступают «коровы» (цифра угадана, но её позиция - нет) и «быки» (когда совпадает и цифра и её позиция). То есть если загадано число «1234», а вы называете «6531», то результатом будет 1 корова (цифра «1») и 1 бык (цифра «3») .

Попытки отгадать число будут идти через диалоговое окно – prompt. Браузер будет сообщать в ответ больше или меньше загаданного наше предположение.

Алгоритм будет таким:

1. Браузер генерирует число и приглашает пользователя к игре.
2. Выводится окно запроса предположения.
3. Браузер проверяет число и возвращает результат.
4. Повторяем до тех пор, пока число не будет угадано.
5. Как только число угадано, браузер сбрасывает число попыток и генерирует новое число.

Домашнее задание

1. С помощью цикла while вывести все простые числа в промежутке от 0 до 100

2. Начиная с этого урока, мы начинаем работать с функционалом интернет-магазина. Предположим, что у нас есть сущность корзины. Нужно реализовать функционал подсчета стоимости корзины в зависимости от находящихся в ней товаров. Товары в корзине хранятся в массиве.

2.1. Организуйте такой массив для хранения товаров в корзине

2.2. Организуйте функцию countBasketPrice, которая будет считать стоимость корзины.

3. *Вывести с помощью цикла for числа от 0 до 9, НЕ используя тело цикла. То есть выглядеть должно вот так:

for(…){// здесь пусто}

4. * Нарисовать пирамиду с помощью console.log, как показано на рисунке, только у вашей пирамиды должно быть 20 рядов, а не 5:

```
x

xx

xxx

xxxx

xxxxx
```

Дополнительные материалы

1. http://www.tvd-home.ru/recursion - про рекурсию
2. http://devenergy.ru/archives/category/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D1%8B-%D0%BE%D1%81%D0%BD%D0%BE%D0%B2%D1%8B - про алгоритмы

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. «JavaScript. Подробное руководство» - Дэвид Флэнаган
2. «Изучаем программирование на JavaScript» - Эрик Фримен, Элизабет Робсон