Урок 3

Циклы, массивы, структуры данных

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

Понятие цикла

Циклы в JavaScript

- Цикл while
- Цикл do..while
- Цикл for
- Цикл for/in
- Бесконечный цикл и выход из шагов цикла

Массивы

Практикум

Практическое задание

Дополнительные материалы

Используемая литература

Понятие цикла

Цикл — разновидность управляющей конструкции в высокоуровневых языках программирования, организует многократное исполнение набора инструкций. Цикл заставляет группу действий повторяться до наступления нужного условия. К примеру:

- опрашивать клиентов, пока не наберем сто ответов;
- читать строчки из файла, пока не доберемся до его конца.

Набор инструкций называется телом цикла. Каждое выполнение тела цикла — это итерация. Условие выхода определяет, делать ли следующую итерацию или завершить цикл. Удобно знать, сколько итераций уже сделано — то есть хранить их число в переменной. Она называется счетчиком итераций цикла и не обязательна для циклов.

Цикл состоит из следующих шагов:

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

В большинстве языков программирования есть инструменты досрочного прерывания всего цикла или текущей итерации.

Циклы в JavaScript

В JavaScript есть четыре цикла: **while**, **do/while**, **for** и **for/in**. Каждому из них посвящен один из следующих разделов. Одно из обычных применений циклов — обход элементов массива.

Цикл while

Цикл **while** — пример цикла с предусловием. Он выполняется, пока истинно условие, указанное перед его началом. Поскольку условие проверяется до выполнения первой итерации, вполне может не выполниться ни одной, 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: например, значение **boolean**, которое обращается в **false** при достижении граничного условия, или целочисленное значение, которое каждый раз увеличивается на единицу, пока не достигнет определенного значения.

На практике:

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

console.log() — удобная функция, чтобы выводить отладочную информацию, не мешая пользователю всплывающими сообщениями. Посмотреть ее можно в отладчике браузера на вкладке Console. Обратите внимание, что данная функция совместима с IE от восьмой версии и выше.

В начале переменной **count** присваивается значение 0, а затем оно увеличивается каждый раз, когда выполняется тело цикла. Когда цикл будет выполнен десять раз, выражение вернет **false** (значит переменная **count** уже не меньше 10). Инструкция **while** завершится, и интерпретатор перейдет к следующей инструкции в программе. Большинство циклов имеют переменные-счетчики, аналогичные **count**. Чаще всего в качестве счетчиков цикла выступают переменные с именами **i**, **j** и **k**, хотя чтобы сделать программный код понятнее, следует давать счетчикам более «говорящие» имена.

Цикл do..while

do…while — это цикл с постусловием, работающий по такому алгоритму:

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

В цикле с постусловием условие проверяется после выполнения тела. Следовательно, тело всегда выполняется хотя бы один раз.

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

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

Цикл **do...while** используется достаточно редко из-за его громоздкости и плохой читаемости. Да и ситуация, когда необходимо гарантировать хотя бы одно выполнение тела цикла, — экзотичная.

Цикл for

for — цикл со счетчиком. Сам по себе он представляет пример лаконичной организации кода, имея максимально схожий вид в большинстве языков программирования.

for значительно упрощает объявление циклов. Как мы видели, даже без прямого требования циклы имеют счетчик. Он объявляется перед началом цикла и проверяется на каждой итерации, изменяя значение внутри тела цикла согласно правилам, установленным программистом. Инициализация, проверка и обновление — три ключевых операции, которые выполняют со счетчиком. Цикл **for** объединяет эти три шага:

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

Если провести аналогию с циклом **while**, получится такой псевдокод:

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

В цикле **for** перед его началом выполняется инициализация, в которой обычно и объявляется переменная-счетчик. Проверка вычисляется в начале каждой итерации цикла. Если она возвращает **true**, цикл продолжается, в ином случае останавливается. По окончании итерации производится инкрементирование счетчика.

Выведем в цикле 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**. Рассмотрим синтаксис цикла **for/in**:

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

В параметре «переменная» обычно используется имя переменной, но можно использовать и инструкцию **var**, объявляющую единственную переменную. Параметр «объект» — это выражение, возвращающее объект. А инструкция — это одна инструкция или их блок, образующий тело цикла.

Цикл **for/in** позволяет обходить объекты:

```
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

Вспомним игру с прошлого занятия. Чтобы решить вопрос с тем, что постоянно запрашивались данные у пользователя, приходилось применять рекурсию. На самом деле это неправильно, так как память расходовалась не оптимально. Более изящное и простое решение — применить бесконечный цикл.

Бесконечный цикл — бич неопытных программистов. Если в него войти без соответствующего контроля, то с каждой итерацией он будет потреблять больше и больше памяти, и в конечном итоге «повесит» браузер пользователя. Но в умелых руках бесконечный цикл можно сделать полезным.

Бесконечным считается цикл такого вида:

```
while(true){ ... }
```

Возникает вопрос, как ими управлять, если условие выхода никогда не вернет **false**? Для выхода из всего цикла используется оператор **break**.

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

Здесь выводим числа от 1 до 10 при помощи бесконечного цикла. Каждый раз в конце цикла проверяется состояние переменной **i**. Как только она станет равна 11, вызовется оператор **break**, который мгновенно завершит выполнение цикла и приступит к инструкциям, идущим после объявления цикла в коде.

Не всегда нужно останавливать весь цикл. Иногда требуется 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++);
}
```

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. Его можно представить в виде большого шкафа со множеством ящиков. Сам шкаф — это массив, а ящики в нем — элементы массива.

В классическом виде нумерация элементов (ящиков) начинается с нуля. Ключи не могут повторяться — они уникальны. Так что массив можно определить как переменную, которая может хранить не одно, а множество значений.

Отсчет индексов массивов начинается с нуля, и для них используются 32-битные целые. Массивы в JavaScript динамические: они могут увеличиваться и уменьшаться. 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. Это простой список разделенных 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 создавать массивы — вызывать конструктор **Array()**. Это 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 свойство размера — **length**, которое подсчитывается автоматически. Это может пригодиться при обходе массива в цикле.

Наиболее простой способ добавлять элементы в массив — присваивать значения по индексам. Чтобы добавить один или несколько 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
```

Чтобы вставить элемент в начало массива, можно использовать метод **unshift()**. Но при этом существующие элементы изменят свои индексы и сменят позиции.

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

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

При удалении одного элемента другие не смещаются — удаленному индексу присваивается значение **empty**.

У массивов есть метод **pop()**, противоположный **push()**. Он уменьшает длину массива на единицу и возвращает значение удаленного элемента. Метод **shift(),** противоположный **unshift()**, удаляет элемент в начале массива. В отличие от оператора **delete**, метод **shift()** сдвигает все элементы на позицию ниже их текущих индексов.

Чтобы обработать каждый элемент, выполняют 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[i]);
}
```

Для многоуровневых массивов можно применять вложенные циклы, если на момент старта точно известен уровень вложенности. В иных случаях нужно применять рекурсивные функции.

Быки и коровы

Усложним предыдущую игру. Соперник (компьютер, например), загадывает четырехзначное число, в котором не повторяются цифры. Ваша задача — угадать его. Число ходов можно ограничить. Подсказками выступают «коровы» (если цифра угадана, а ее позиция — нет) и «быки» (когда совпадает и цифра, и ее позиция). Если загадано число «1234», а вы называете «6531» — результатом будет одна корова (цифра «1») и один бык (цифра «3»).

Попытки отгадать число идут через диалоговое окно — **prompt**. Браузер сообщает, больше или меньше загаданного числа предложенный вариант.

Алгоритм будет таким:

1. Браузер генерирует число и приглашает пользователя в игру.
2. Выводится окно запроса предположения.
3. Браузер проверяет число и возвращает результат.
4. Так повторяется до тех пор, пока число не угадают.
5. Как только это случается, браузер сбрасывает количество попыток и генерирует новое число.

Практическое задание

1. С помощью цикла **while** вывести все простые числа в промежутке от 0 до 100.
2. С этого урока начинаем работать с функционалом интернет-магазина. Предположим, есть сущность корзины. Нужно реализовать функционал подсчета стоимости корзины в зависимости от находящихся в ней товаров.
3. Товары в корзине хранятся в массиве. Задачи:
    a. Организовать такой массив для хранения товаров в корзине;
    b. Организовать функцию **countBasketPrice**, которая будет считать стоимость корзины.
4. * Вывести с помощью цикла **for** числа от 0 до 9, не используя тело цикла. Выглядеть это должно так:

```
for(...){// здесь пусто}
```

5. * Нарисовать пирамиду с 20 рядами с помощью **console.log**, как показано на рисунке:

```
x
xx
xxx
xxxx
xxxxx
```

Дополнительные материалы

1. Рекурсия и рекурсивные алгоритмы.
2. Блог о программировании. Алгоритмы.

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

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