УВАЖАЕМЫЕ СТУДЕНТЫ! Законспектируйте в своей рабочей тетради по дисциплине приведенную лекцию (объемом 4-5 страницы), ответьте *письменно* на контрольные вопросы.

Результаты работы, фотоотчет, предоставить преподавателю на e-mail: igor-gricenko-95@mail.ru **в течении ТРЕХ дней.**

При возникновении вопросов по приведенному материалу обращаться по следующему номеру телефона: (072)132-63-42

ВНИМАНИЕ!!! При отправке работы, не забывайте указывать ФИО студента, наименование дисциплины, дата проведения занятия (по расписанию).

Лекция 23

Тема: Операторы цикла, массивы в JavaScript (продолжение)

Цель: Изучить основные операторы цикла и массивы в JavaScript

1.1 Массивы

Массивы делятся на встроенные (document.links[], document.images[] и т.п. — их еще называют *коллекциями*) и определяемые пользователем (автором документа). Коллекции будут обсуждаться в следующей лекции. Здесь же мы подробно остановимся на массивах, определяемых пользователем. Для массивов определено несколько методов: join(), reverse(), sort() и другие, а также свойство length, которое позволяет получить число элементов массива.

Для определения массива пользователя существует специальный конструктор Array. Если ему передается единственный аргумент, причем целое неотрицательное число, то создается незаполненный массив соответствующей длины. Если же передается один аргумент, не являющийся числом, либо более одного аргумента, то создается массив, заполненныйэтими элементами:

```
a = new Array();
// пустой массив (длины 0)
b = new Array(10);
// массив длины 10
c = new Array(10,'Привет');
// массив из двух элементов: числа и строки
d = [5, 'Тест', 2.71828, 'Число e'];
// краткий способ создать массив из 4 элементов
```

Элементы массива нумеруются с нуля. Поэтому в последнем примере значение d[0] равно 5, а значение d[1] равно 'Тест'. Как видим, массив может состоять из разнородных элементов. Массивы не могут быть многомерными, однако ничтоне мешает завести массив, элементами которого будут тоже массивы.

Метод join()

Метод join() позволяет объединить элементы массива в одну строку. Он является обратным к методу split(), который разрезает объект типа String на куски и составляет из них массив. Кстати, метод split()демонстрирует тот факт, что массив можно получить и без конструкторамассива.

Рассмотрим пример преобразования локального URL в глобальный URL, где в качестве адреса сервера будет выступать its.kpi.ua. Пусть в переменной localURLхранится локальный URL некоторого файла:

localURL = "file:///D:/courses/internet/js/2/2.html"

Разрежем строку в местах вхождения комбинации символов ":/", выполнив команду:
b = localURL.split(':/')

Получим массив:
b[0] = "file";

b[1] = "//D";

b[2] = "courses/internet/js/2/2.html";

Заменяем 0-й и 1-й элементы на требуемые:
b[0] = "http:";

b[1] = "/its.kpi.ua";

Наконец, склеиваем полученный массив, вставляя косую черту в местах склейки: globalURL = b.join("/"). В итоге мы получаем требуемый глобальный URL — значение globalURLбудет равно:

http://its.kpi.ua/courses/internet/js/2/2.html.

Метод reverse()

Метод reverse() применяется для изменения порядка элементов массива на противоположный. Предположим, массив упорядочен следующимобразом:

a = new Array('мать','видит','дочь');

Упорядочим его в обратном порядке, вызвав метод
a.reverse()

Тогда новый массив абудет содержать:
a[0]='дочь';

a[1]='видит';

a[2]='мать';

Метод sort()

Метод sort() интерпретирует элементы массива как **строковые литералы** и сортирует массив в **алфавитном** (т.н. лексикографическом) порядке. Обратите внимание: метод sort() меняет массив. В предыдущем примере, применив

a.sort()

мы получим на выходе:
a[0]='видит';

a[1]='дочь';

a[2]='мать';

Однако, это неудобно, если требуется отсортировать числа, поскольку согласно алфавитному порядку 40 идет раньше чем 5. Для этих целей у метода sort() имеется необязательный аргумент, являющийся именем функции, согласно которой требуется отсортировать массив, т.е. в этом случае вызов метода имеет вид: a.sort(myfunction). Эта функция должна удовлетворять определенным требованиям:

- у нее должно быть ровно два аргумента;
- функция должна возвращать число;
- если первый аргумент функции должен считаться меньшим (большим, равным) чем второй аргумент, то функция должна возвратить

отрицательное (положительное, ноль) значение.

Например, если нам требуется сортировать числа, то мы можем описать следующую функцию:

```
function compar(a,b)
{
if(a < b) return -1;
if(a > b) return 1; if(a ==
b) return 0;
}
```

Теперь, если у нас есть массив
b = new Array(10,6,300,25,18);

то можно сравнить результаты сортировки без аргумента и с функцией comparв качестве аргумента:

```
document.write("Алфавитный
порядок:<BR>"); document.write(b.sort());
document.write("<BR>Числовой
порядок:<BR>");
document.write(b.sort(compar));
```

В результате выполнения этого кода получим следующее:
Алфавитный порядок:

10,18,25,300,6

Числовой порядок:

6,10,18,25,300

Обратите внимание: метод sort() **интерпретирует** элементы массива как строки (и производит лексикографическую сортировку), но не **преобразует** их в строки. Если в массиве были числа, то они числами и останутся. В этом легко убедиться, если в конце последнего примера выполнить команду document.write(b[3]+1): результат будет 26 (т.е. 25+1), а не 251(т.е. "25"+1).

1.2 Операторы языка

В этом разделе будут рассмотрены операторы JavaScript. Основное внимание при этом мы уделим операторам декларирования и управления

потоком вычислений. Без них не может быть написана ни одна JavaScript-
программа.

Общий перечень этих операторов выглядит следующим образом (сразу оговоримся, что этот список неполный):

- {...}
- if ... else ...
- ()?
- while
- for
- break
- continue
- return

Блок: {...}

Фигурные скобки определяют составной оператор JavaScript — ***блок***. Основное назначение блока — определение тела цикла, тела условного оператора или функции.

Условный оператор: if ... else ...

Условный оператор применяется для ветвления программы по некоторому логическомуусловию. Есть два варианта синтаксиса:

if (логическое_выражение) оператор;

if (логическое_выражение) оператор_1; else оператор_2;

Логическое выражение — это выражение, которое принимает значение true или false. В первом варианте синтаксиса: если логическое_выражение равно true, то выполняется указанный оператор. Во втором варианте синтаксиса: если логическое_выражение равно true, то выполняется оператор_1, если же оно равно falseоператор_2.

Пример использования (об объекте navigatorчитай лекцию 4):
if (navigator.javaEnabled())

alert('Ваш браузер поддерживает
Java');else

alert('Ваш браузер НЕ поддерживает Java');

Тернарная условная операция: ()?

Этот оператор, называемый *условным выражением*, выдает одно из двух значений взависимости от выполнения некоторого условия. Синтаксис его таков:

(логическое_выражение)? значение_1 : значение_2

Если логическое_выражение равно true, то возвращается значение_1, в противном случае значение_2. Условное выражение легко имитируется оператором if...else, однако оно позволяет сделать более компактным и легко воспринимаемым код программы. Например, следующие два фрагмента равносильны:

```
TheFinalMessage = (k>5)? 'Готово!' : 'Подождите...';
```

```
if(k>5) TheFinalMessage = 'Готово!';
else TheFinalMessage = 'Подождите...';
```

Оператор цикла while

Оператор whileзадает цикл. Определяется он в общем случае следующим образом:

while (условие_продолжения_цикла) тело_цикла;

Тело цикла может быть как простым, так и составным оператором. Составной оператор, как всегда, заключается в фигурные скобки.Рекомендуется и простой оператор заключать в них, чтобы программу можно было легко модифицировать. Условие_продолжения_цикла является логическим выражением. Тело исполняется до тех пор, пока верно логическое условие. Формально, цикл whileработает следующим образом:

1) проверяется условие_продолжения_цикла:
   a) если оно ложно (false), цикл закончен,
   b) если же истинно (true), то продолжаем далее;
2) выполняется тело_цикла;

3) переходим к пункту 1.

Такой цикл используется, когда заранее неизвестно количество итераций, например, в ожидании некоторого события. Пример:

```
var s='';
while (s.length<6)
{
s=prompt('Введите строку длины не менее 6:','');
}
alert('Ваша строка: ' + s + '. Спасибо!');
```

Оператор цикла for

Оператор for— это еще один оператор цикла. В общем случае он имеет вид:

for (инициализация_переменных_цикла;
условие_продолжения_цикла;
модификация_переменных_цикла) тело_цикла;

Тело цикла может быть как простым, так и составным оператором (составной необходимо заключать в фигурные скобки). Операторы инициализация_переменных_цикла и модификация_переменных_цикла могут состоять из нескольких простых операторов, в этом случае простые операторы должны быть разделены **запятой**. Условие_продолжения_цикла является логическим выражением. Цикл for работает следующим образом:

1) выполняется инициализация_переменных_цикла;
2) проверяется условие_продолжения_цикла:
a) если оно ложно (false), цикл закончен,
b) если же истинно (true), то продолжаем далее;
3) выполняется тело_цикла;
4) выполняется модификация_переменных_цикла;
5) переходим к пункту 2.

Рассмотрим типичный пример использования этого оператора:

```
document.write('Кубы чисел от 1 до
100:');for (n=1; n<=100; n++)
```

```
document.write('<BR>'+n+'<sup>3</sup> = '+ Math.pow(n,3));
```

Здесь Math — встроенный объект, предоставляющий многочисленные математические константы и функции, а Math.pow(n,m) вычисляет степенную функцию n^m.

Оператор выхода из цикла break

Оператор break позволяет досрочно покинуть тело цикла. Возвращаясь к нашему примеру с кубами чисел, распечатаем только кубы, не превышающие 5000.

```
document.write('Кубы чисел, меньшие 5000:');


for (n=1; n<=100; n++)
{
s=Math.pow(n
,3); if(s>5000) break;


document.write('<BR>'+n+'<sup>3</sup> = '+s);
}
```

Несмотря на то, что переменную nмы заставили пробегать от 1до 100, т.е. заведомо с запасом, реально же цикл выполнится для значений nот 1 до

... получите сами!

Оператор перехода к следующей итерации цикла continue

Оператор continue позволяет перейти к следующей итерации цикла, пропустив выполнение всех нижестоящих операторов в теле цикла. Если нам нужно вывести кубычисел от 1до 100, превышающие 10 000, то мы можем составить такой цикл:

```
document.write('Кубы чисел от 1 до 100, большие 10 000:');
for (n=1; n<=100; n++)
{
s=Math.pow(n,3);
```

```
if(s <= 10000) continue;
document.write('<BR>'+n+'<sup>3</sup> = '+s);
}
```

Разумеется, для большей гибкости можно использовать в циклах оба оператора breaki continue.

Оператор возврата значения из функции return

Оператор return используют для возврата значения из *функции* или обработчика события. Рассмотрим пример с функцией:

```
function sign(n)
{
if (n>0) return 1;
if (n<0) return -1;return
0;

}
alert( sign(-3) );
```

Обратите внимание: оператор return не только указывает, какое значение должна вернуть функция, но и прекращает выполнение дальнейших операторов в теле функции.

При использовании в обработчиках событий оператор return позволяет отменить или не отменять действие по умолчанию, которое совершает браузер при возникновении данного события. Отменить его, однако, можно не для всех событий. Рассмотрим пример:

```
<FORM ACTION="newpage.html" METHOD=post>
<INPUT TYPE=submit VALUE="Отправить?"
onClick="alert('Не отправим!');return false;">
</FORM>
```

В этом примере без оператора return false пользователь увидел бы окно предупреждения "Не отправим!" и далее был бы перенаправлен на страницу newpage.html. Оператор же return false позволяет отменить отправку формы, и пользователь лишь увидит окнопредупреждения.

Аналогично, чтобы отменить действие по умолчанию для параметров событий onClick, onKeyDown, onKeyPress, onMouseDown, onMouseUp, onSubmit, onReset, нужно использовать return false.

Для события onMouseOver с этой же целью нужно использовать оператор return true. Для некоторых же событий, например onMouseOut, onLoad, onUnload, отменить действие по умолчанию невозможно.