

**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](mailto: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 21-22 (продолжение)

**Тема: Выражения, функции, операции и условия в JavaScript**

**Цель: Изучить приемы работы с выражениями, функциями, операциями и условиями в JavaScript**

**Операторы, выражения, функции**

#### **1. Операторы: арифметических действий, присваивания, инкрементные, декрементные. Условные выражения**

Операторы арифметических действий в JavaScript те же, что и в С и Java:

Сложение "+", вычитание "-", умножение "\*", деление "/", остаток от целочисленного деления "%".

Эти операторы могут встречаться в любом численном выражении. (Внимание! Они также могут встречаться в строковых выражениях в случаях, когда используется автоматическое приведения чисел в строки ).

Операторы присваивания в JavaScript те же, что и в С и Java:

"=", "+=", "-=", "\*=", "/=", "%="

$x=y$ , как и в большинстве других языков, значит присвоить переменной "x" значение переменной "y".

Следующие операторы имеют синтаксис, сходный с синтаксисом соответствующих операторов языка С:

$y+=x$  эквивалентно  $y=y+x$

$y-=x$  эквивалентно  $y=y-x$

$y*=x$  эквивалентно  $y=y*x$

$y/=x$  эквивалентно  $y=y/x$

$y\%=x$  эквивалентно  $y=y\%x$  – остаток от деления нацело  $y$  на  $x$ .

Условное выражение имеет вид

$(\text{условие})?\text{значение1}:\text{значение2}$

Если значение условия `true`, значением условного выражения будет `значение1`, иначе — `значение2`. Условное выражение можно применять везде, где можно применять обычные выражения.

Пример:

$a=(b<1)?0:(x-1)+c$

Инкрементные и декрементные операторы также имеют синтаксис, заимствованный из языка C: `"x++"`, `"++x"`, `"x--"`, `"--x"`.

Выражения:

$y=x++$  эквивалентно двум присваиваниям:  $y=x$ ;  $y=y+1$ ,

$y=++x$  эквивалентно двум присваиваниям:  $x=x+1$ ;  $y=x$ ,

$y=x--$  эквивалентно двум присваиваниям:  $y=x$ ;  $x=x-1$ ,

$y=--x$  эквивалентно двум присваиваниям:  $x=x-1$ ;  $y=x$ .

## 2. *Строковые операции*

Существуют ряд операторов работы со строками:

`"+"` - сложение (конкатенация) строк  $s1+s2$  дает строку, состоящую из последовательности символов строки  $s1$ , за которыми следуют символы строки  $s2$ .

`eval(s)` - встроенная функция JavaScript. Она выполняет код, заданный аргументом— строкой  $s$ , который может содержать один или более операторов JavaScript (через точки с запятой). Данную функцию можно использовать не только для выполнения оператора, но и для вычисления выражения. Она возвращает значение последнего вычисленного выражения в заданном коде. Функция `eval(s)` обеспечивает возможность вычислять значения, введенные пользователем в пункты ввода, а также динамически модифицировать выполняемый код в JavaScript-программе. Более общая, чем функции `parseInt` и `parseFloat`.

`parseFloat(s)` – встроенная функция JavaScript. Находит содержащееся в строке `s` вещественное число (типа `Float`) от начала строки до первого символа, не входящего в число. Если число не найдено, возвращает значение `NaN` (“Not a Number”)

`parseInt(s)` – то же для целых чисел (`Integer`). При этом автоматически находится основание.

`parseInt(s,n)` – то же для целых чисел по основанию `n` (от 2 до 36). При `n=0` – то же, что `parseInt(s)`. При этом автоматически находится основание

### 3. Побитовые операции присваивания

Существуют ряд операторов побитового присваивания:

$x \ll= n$  эквивалентно  $x = (x \ll n)$  — сдвиг на `n` битов влево двоичного представления целого числа `x`;

$x \gg= n$  эквивалентно  $x = (x \gg n)$  — сдвиг на `n` битов вправо двоичного представления целого числа `x` с сохранением знакового бита (отрицательные числа в дополнительном коде имеют первым битом единицу. После сдвига на место первого бита записывается 1, и число остается отрицательным);

$x \ggg= n$  эквивалентно  $x = x \ggg n$  — сдвиг на `n` битов вправо двоичного представления целого числа `x` с ведущим нулем 0 (После сдвига на место первого бита записывается 0, и число становится положительным);

Всё выражение (у нас — `"x"`) в левой части преобразуется в 32-битное целое число, после чего производится необходимый сдвиг, а затем получившееся число приводится к типу выражения – результата (у нас это опять `"x"`), и производится присваивание.

Примеры:

1)  $9 \ll 2$  дает 36, т.к. сдвиг 1001 (число 9 в двоичном представлении) на 2 бита влево

дает 100100, т.е. 36. Недостающие биты заполняются 0.

$9 \gg 2$  дает 2, т.к. сдвиг 1001 (число 9 в двоичном представлении) на 2 бита вправо

дает 10, т.е. 2. Недостающие биты заполняются 0.

"&" — побитовая AND — "И";

"|" — побитовое OR — "ИЛИ";

"^" — побитовое XOR — "ИСКЛЮЧАЮЩЕЕ ИЛИ".

Все операции совершаются с двоичным представлением чисел, однако, результат возвращается в обычном десятичном представлении.

Примеры:

$15 \& 9$  возвращает 9, т.к. (1111) AND (1001) равно 1001;

$15 | 9$  возвращает 15, т.к. (1111) OR (1001) равно 1111;

$15 \wedge 9$  возвращает 6, т.к. (1111) XOR (1001) равно 0110.

2.3 Логические выражения.

"&&" — логическое AND — "И";

"||" — логическое OR — "ИЛИ";

"!" — логическое NOT — "НЕ".

Пример:

$(a > b) \&\&(b \leq 10) || (a > 10)$

В JavaScript всегда применяется так называемая сокращенная проверка логических выражений: в операнде  $B1 \&\&B2$  при  $B1 = \text{false}$  оценки  $B2$  не производится и возвращается значение `false`. Аналогично  $B1 || B2$  при  $B1 = \text{true}$  оценивается как `true`. При этом анализ логического выражения идет слева направо, и как только всё выражение может быть оценено, возвращается результат. Поэтому, если в качестве  $B2$  выступает функция, она не будет вызываться, и если она должна давать побочные эффекты, то это может привести к ошибке.

#### 4. Операторы сравнения

"=" - "равно";

">" - "больше";

"<" - "меньше";

">=" - "больше или равно";

"<=" - "меньше или равно";

"!=" - "не равно".

Операторы сравнения применимы не только к численным, но и к строковым выражениям. При этом строки считаются равными, если все их символы совпадают и идут в одном и том же порядке (пробел учитывается как символ). Если строки разной длины, то большей будет строка имеющая большую длину. Если их длины совпадают, больше считается та, у которой при просмотре слева направо встретится символ с большим номером ( $a < b < c < \dots < z < A < \dots < Z$ ).

Строки можно складывать, если  $S1 = \text{"это "}$ ,  $S2 = \text{"моя строка"}$ , то  $S1+S2$  даст  $\text{"это моя строка"}$ .

Приоритет операторов (начиная с младших; в одной строке старшинство одинаково):

"+=", "-=", "\*=", "/=", "%=", "<<=", ">>=", ">>>=", "&=", "^=", "|=".

## 5. *Старшинство операций*

условное выражение: "?:";

логическое "ИЛИ": "||";

логическое "И": "&&";

побитовое "ИЛИ": "|";

побитовое "XOR": "^";

побитовое "И": "&";

сравнение на равенство "=", "!=";

сравнение: "<", "<=", ">", ">=";

побитовый сдвиг: "<<", ">>", ">>>";

сложение, вычитание: "+", "-";

умножение, деление: "\*", "/";

отрицание, инкремент, декремент: "!", "~", "++", "--";

скобки: "()", "[ ]".

## 6. *Функции*

В JavaScript, как и в C или Java, процедуры и процедуры — функции называются функциями. Декларация функции состоит из:

зарезервированного слова `function`;

имени функции;

списка аргументов, разделенных запятыми, в круглых скобках;

тела функции в фигурных скобках.

```
function myFunction(arg1, arg2, ...)
```

```
{
```

```
...
```

последовательность операторов

```
...
```

```
}
```

где `myFunction` — имя функции, `arg1, arg2` — список формальных параметров

Пример:

```
function Factorial(n)
```

```
{
```

```
if((n<0)||((round(n)!=n))
```

```
{
```

```
alert("функция Factorial не определена при аргументе "+n);
```

```
return NaN;
```

```
}
```

```
else
```

```
{
```

```
result=(n*Factorial(n-1));
```

```
return result;
```

```
}
```

```
}
```

Функция может не возвращать значения с помощью зарезервированного слова `return`.

Пример:

```
function Greeting(s)
{ document.write("Hello,"+s+"!");
}
```

Вызов функции производится с фактическими параметрами.

Пример:

```
Factorial(2+1);
```

— внутри некоего выражения возвратит 6, а

```
Greeting("world");
```

—приведет к выводу на экран строки "Hello, world!".

Каждая функция, например `myFunction`, является объектом с именем `myFunction`, аргументы которого хранятся как массив, имеющий имя `arguments`, при этом доступ к аргументам может осуществляться так:

`myFunction.arguments[i]`, где `i` — номер аргумента (нумерация начинается с 0).

Число фактических параметров должно быть равно либо больше числа формальных параметров в описании функции. При этом, при вызове функции действительное число переданных аргументов хранится в поле `myFunction.arguments.length` и может динамически изменяться переприсваиванием значения этого поля.

Пример:

Вывод в документе списка в формате HTML.

Первый аргумент здесь (`ListType`) может иметь значение "o" или "O" для упорядоченного списка и "u" или "U" для неупорядоченного. Далее идут элементы списка.

```
function myList(ListType)
{
document.write("<"+ListType+"L");
```

```
for(var i=1; i < myList.arguments.length; i=i+1)
{ document.write("<LI>" +myList.arguments[i]);
}
document.write("</"+ListType+"L>");
}
```

Вызов в тексте HTML документа этой функции:

```
<script>
myList("0", "один", 2, "3")
</script>
```

приведет к выводу текста:

```
один
2
3
```

## 7. Условный оператор *if*

первый вариант синтаксиса оператора *if*:

```
if(условие)
{
утверждение
}
```

(Утверждением называется оператор или последовательность операторов.)

b) второй вариант синтаксиса оператора *if*:

```
if(условие)
{
утверждение1
}
else
{
утверждение2
}
```



Пример использования условного оператора:

```
function checkData()
{
if (document.form1.threeChar.value.length==3)
{return true;
}
else
{alert('Введите ровно 3 символа');
return false;
}
}
```

## 8. Цикл *for*

```
for(секция инициализации; секция условия; секция изменения счетчиков)
{
утверждение
}
```

Каждая из секций может быть пустой. В секциях инициализации и изменения счетчиков можно писать последовательности выражений, разделяя их запятыми. Выполнение цикла происходит следующим образом. Первой выполняется секция инициализации. Затем проверяется условие. Если условие равно true, то выполняется тело цикла, а затем секция изменения счетчиков. Если условие равно false, то выполнение цикла прекращается.

Пример :

```
function HowMany(SelectObject)
{
var numberSelected=0
for (i=0; i< SelectObject.options.length; i++)
{
if (SelectObject.options[i].selected==true) numberSelected++;
}
```

```
}  
return numberSelected;  
}
```

Оператор for может использоваться для перебора всех полей объекта (см. далее раздел про объектную модель)

Синтаксис:

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

При этом производится перебор всех возможных значений указанной переменной в объекте, и для каждого из них выполняется утверждение.

Пример:

```
function student(name, age, group)  
{  
    this.name=name;  
    this.age=age;  
    this.group=group;  
}  
function for_test(myObject)  
{  
    for(var i in myObject)  
    {  
        document.write("i="+i+" => "+myObject[i]+"/n");  
    }  
};  
Helen=new student("Helen K.", 21, 409);  
for_test(Helen);  
Вывод на экран:  
i=0 => Helen K.  
i=1 => 21
```

i=2 => 409

## 9. Цикл *while*

```
while(условие)
{
    выражение
}
```

Выполнение цикла `while` начинается с проверки условия. Если оно равно `true`, то выполняется тело цикла, иначе управление передается оператору, следующему за циклом.

Пример использования оператора `while`:

```
n1=10
n=0
x=0
while(n<n1)
{
    n=n+1;
    x=x+n;
}
```

### 2.8 Операторы прерывания выполнения циклов.

Для прекращения выполнения текущего цикла операторов `for` или `while` служит оператор `break`.

Пример использования оператора `break`:

```
function test(x)
{
    var j=0;
    var sum=0;
    while(n<n1)
    {
        if(n==x)
        { sum=x;
          break;
        }
    }
}
```

```
}  
sum=sum+n;  
n=n+1;  
}  
return sum;  
}
```

Оператор `continue` прерывает выполнение текущей итерации внутри `for` или `while` и вызывает переход к началу следующей итерации.

Пример использования оператора `continue`:

```
function test1(x)  
{  
  var j=0;  
  while(n<n1)  
  { if(n==x)  
    { sum=x;  
      continue;  
    }  
    sum=sum+n;  
    n=n+1;  
  }  
  
  return sum;  
  
}
```