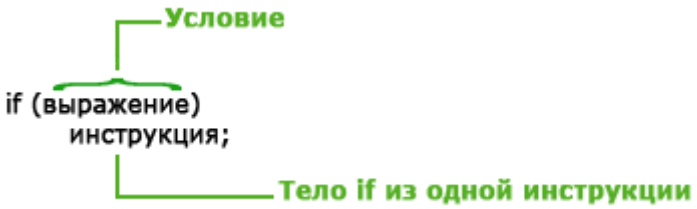
Лекция №2
Условные операторы, функции, циклы

План:

1. Условные операторы
2. Функции
3. Циклы
4. Инструкция switch в JavaScript

Инструкция if

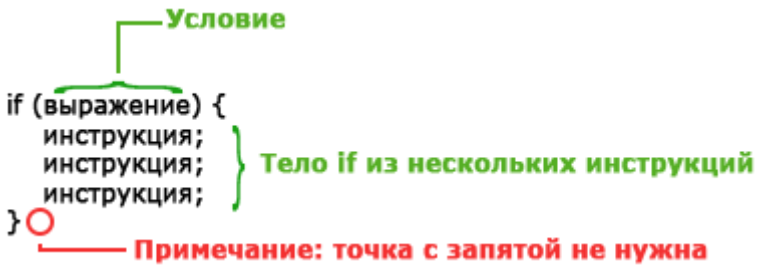
Инструкция if имеет две формы. Первая:



Сначала вычисляется значение выражения. Полученное значение, если необходимо, неявно преобразуется к логическому типу. Само выражение называется условием выполнения инструкции if или просто условие. Если результатом вычисления выражения является истинное значение, то инструкция выполняется. Если выражение возвращает ложное значение, то инструкция не выполняется.

```
var year = prompt('В каком году группа 14ИВТ-4ДБ-011 поступила в институт?', '');
if (year != 2012) alert( 'Неправильно!' );
```

Синтаксис JavaScript позволяет вставить только одну выполняемую инструкцию после инструкции if, однако если требуется выполнить более одной инструкции, её всегда можно заменить блоком инструкций:



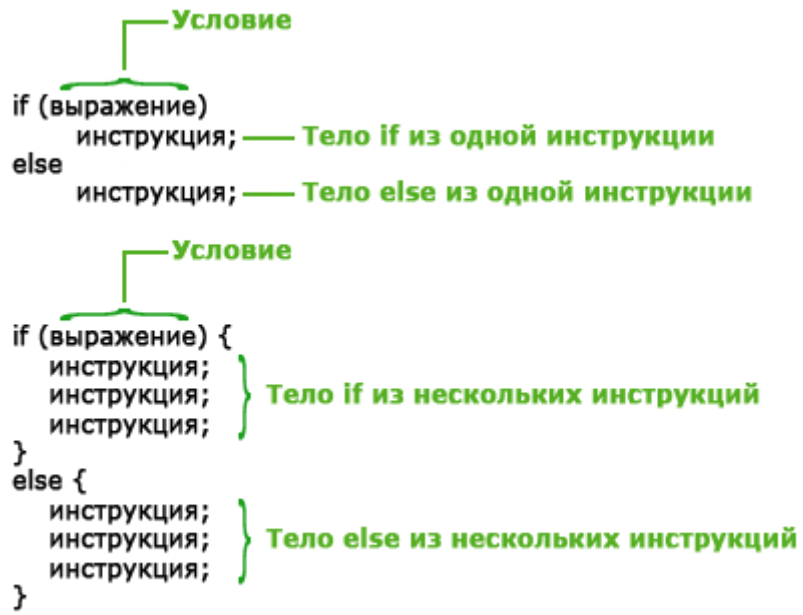
Обратите внимание, даже если используется всего одна инструкция, её можно заключить в фигурные скобки, но это не обязательно:

```
if (year != 2011) {
  alert( 'А вот..' );
  alert( '..и неправильно!' );
}
```

В логическом контексте:

- Число 0, пустая строка "", null и undefined, а также NaN являются false,
- Остальные значения — true.

С инструкцией if может использоваться ключевое слово else, которое позволяет добавить инструкцию (или блок кода), выполняемую, если условие имеет ложное значение. Вторая форма инструкции if:



В этой форме, если выражение возвращает истинное значение выполняется инструкция, расположенная в блоке if, если результатом вычисления является ложное значение, то выполняется инструкция расположенная в блоке else:

```
var num = 15;
if (num > 10)
  alert("число " + num + " больше 10");
else
  alert("число " + num + " меньше 10");
```

Часто на практике бывает необходимо проверить несколько вариантов условий. Для этого можно применить следующую конструкцию:

```
var num = 2;
if (num == 1) {
  alert("значение num: " + num);
} else if (num == 2) {
  alert("значение num: " + num);
} else if (num == 3) {
  alert("значение num: " + num);
} else {
  alert("Не знаю такого числа!");
}
```

2.1 Условный оператор

Оператор	Операция	Типы значений
?:	Выбор второго или третьего операнда	булево, любое, любое→любое

условие ? значение1 : значение2

Условный оператор - это единственный тернарный оператор в JavaScript. Первый операнд предшествует символу ?, второй - между ? и :, третий - после :.

Проверяется условие, затем если оно верно — возвращается значение1, если неверно — значение2, например:

Операнды условного оператора могут быть любого типа. Первый операнд вычисляется и преобразуется (если необходимо) в логическое значение. Он используется в качестве условия - как выражение в круглых скобках инструкции if. **Если первый операнд возвращает истинное значение, то вычисляется и возвращается значение второго операнда. Если первый операнд имеет ложное значение, то вычисляется и возвращается значение третьего операнда.** Обратите внимание, что вычисляется всегда

только значение какого-то одного операнда - второго или третьего, оба никогда не вычисляются.

```
Alert
 (true ? 5 : 10);
```

Условный оператор часто используется как более краткий вариант инструкции if/else.

```
var a, b, num = 2;
if (num)
  a = 5;
else
  a = 10;
```

тоже самое, только с условным оператором

```
var b = num ? 5 : 10;
alert("a: " + a + "<br>");
alert("b: " + b);
```

2.2 Функции

Функция - это блок кода, который определяется один раз и затем может вызываться на выполнение любое количество раз.

Функция определяется с помощью ключевого слова **function**, за которым указываются следующие компоненты:

- Идентификатор, определяющий имя функции.
- Блок кода, заключенный в пару фигурных скобок. Он является телом функции и выполняется при каждом вызове функции.
- Пара круглых скобок, для параметров функции, разделяемых запятыми.

Синтаксис определения функции выглядит следующим образом:

function имя_функции(параметры) { инструкции }

Параметров в функции может быть указано любое количество:

```
function test(a,b,c) { ... }  // a,b,c - параметры функции
```

При вызове функции, ей могут передаваться значения, которыми будут инициализированы параметры. Значения, которые передаются при вызове функции, называются аргументы.

```
function summ(a,b) {
  var c = a + b;
  alert(c);
}
 sum(5, 7);
```

Если при вызове функции ей передаётся больше аргументов, чем задано параметров, "лишние" аргументы просто не присваиваются параметрам. Допустим, имеется следующее определение функции:Когда при вызове функции ей предаётся список аргументов, эти аргументы присваиваются параметрам функции в том порядке, в каком они указаны: первый аргумент присваивается первому параметру, второй аргумент второму параметру и т.д. Если число аргументов отличается от числа параметров, то никакой ошибки не происходит. В JavaScript подобная ситуация разрешается следующим образом:

```
function test(a,b,c) { ... }
```

- Если при её вызове указать test(1,2,3,4,5), то аргументы 1,2 и 3 будут присвоены параметрам a,b и c соответственно. В то же время аргументы 4 и 5 не будут присвоены ни одному из параметров данной функции.
- Если функция имеет больше параметров, чем передано ей аргументов, то параметрам без соответствующих аргументов присваивается значение undefined. Так, если

при вызове функции test(a,b,c) указать test(1), параметру a будет присвоено значение 1, а параметрам b и c - значение undefined.

```
function test(a,b,c) {
    alert(a+" , "+b+" , "+c);
}
test(1);
```

Функция — это действие, поэтому для имён функций, как правило, используются глаголы

Инструкция return

Функция может возвратить некоторое значение (результат работы функции) программе, которая её вызвала. Возвращаемое значение передаётся в точку вызова функции.

Инструкция return внутри функции служит для определения значения, возвращаемого функцией. В качестве возвращаемого значения может быть значение любого типа. Если в качестве возвращаемого значения используется выражение, в программу возвращается не само выражение, а результат его вычисления. Инструкция return имеет следующий синтаксис:

```
return выражение;
```

Для дальнейшего использования возвращаемого значения, результат выполнения функции можно присвоить например переменной:

```
function calc(a) {
  return a * a;
}
 var x = calc(5);
 document.write(x);
```

Инструкция return может быть расположена в любом месте функции. Как только будет достигнута инструкция return, функция возвращает значение и немедленно завершает своё выполнение. Код, расположенный в теле функции после инструкции return, будет проигнорирован.

```
var a = 1;
 function test() {
  ++a;
  return;
  ++a;
}
 test();
 alert(a);  // => 2
```

Внутри функции может быть использовано несколько инструкций return, например, если возвращаемое значение зависит от некоторых условий выполнения:

```
function check(a, b) {
  if(a > b) return a;
  else return b;
}
 document.write(check(3, 5));
```

Если инструкция return не указана или не указано возвращаемое значение, то функция вернёт значение undefined.

Инструкция return не указана

```
function bar() { alert("функция bar выполнилась"); }
```

Возвращаемое значение не указано

```
function test(a) {
  if(!a) return;
  alert(a);
```

```
}
var a = bar(); // undefined
var b = test(); // undefined
 alert("a: " + a + ",   b: " + b);
```

Определение функции неявно "поднимается" в начало сценария или вмещающей её (внешней) функции, благодаря чему функцию можно вызвать ещё до того, как она будет определена.

```
test();
bar();
function test() {
  alert("Hello!");
}
  function bar() {
  var str = "Hello again!";
  test();
 function test() {
    alert(str);
  }
}
```

Таким образом функции выше эквивалентны определениям функций приведённым ниже, в которых определения "подняты" в начало.

```
function test() {
  alert("Hello!");
}
 function bar() {
  function test() {
    document.write(str);
  }
   var str = "Hello again!";
  test();
}
 test();
bar();
```

Функция isNaN().

Таким образом функции выше эквивалентны определениям функций приведённым ниже, в которых определения "подняты" в начало.

Она возвращает true, если аргумент имеет значение NaN или если аргумент является нечисловым значением, таким как строка или объект.

Синтаксис:

```
isNaN(testValue);
    alert(isNaN("123"));          // false
    alert(isNaN(-1.23));          // false
    alert(isNaN("5"-2));          // false
    alert(isNaN(0));             // false
    alert(isNaN("Hello"))
```

Метод parseFloat

Анализирует строковый аргумент и возвращает число с плавающей точкой.

Синтаксис:

```
parseFloat(string)
```

string строка, представляющая собой значение, которое вы хотите проанализировать.

Если аргумент не может быть преобразован в число - возвращает NaN

```
parseFloat("3.14"); // 3.14
parseFloat("тест") // NaN
```

Метод parseInt

Анализирует строковый аргумент и возвращает целое число, определенное как основание.

Синтаксис:

```
parseInt(string [,radix])
```

string строка, которая представляет собой значение, которое вы хотите проанализировать.

radix целое число, представляющее собой основание, возвращаемого значения.

parseInt округляет дробные числа, т.к останавливается на десятичной точке.

Если radix не указан или равен 0, то javascript предполагает следующее:

- Если входная строка начинается с "0x", то radix = 16
- Если входная строка начинается с "0", то radix = 8. Этот пункт зависит от реализации и в некоторых браузерах (Google Chrome) отсутствует.
- В любом другом случае radix=10.

```
parseInt(" 0xF", 16)
parseInt(" F", 16)
parseInt("17", 8)
parseInt(021, 8)
parseInt("015", 10)
parseInt(15.99, 10)
parseInt("FXX123", 16)
parseInt("1111", 2)
```

Если преобразовать в число не удается, parseInt возвращает NaN

Пример: все вызовы вернут NaN

```
parseInt("Hello", 8); // вообще не число
parseInt("546", 2);   // цифры не входят в 2-чную систему
```

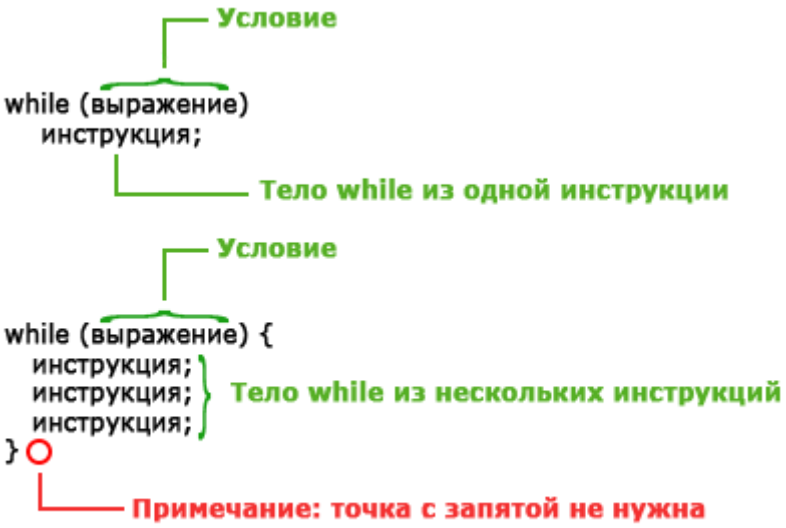
2.3 Циклы

Цикл - это инструкция, позволяющая повторять выполнение некоторых действий (инструкций) определённое количество раз. Каждое отдельное исполнение инструкций в теле цикла называется итерация

Каждый цикл состоит из двух основных частей. Первая определяет, условие выполнения цикла. Вторая - собственно фрагмент программного кода, выполняющий необходимые действия, который может состоять из одной единственной инструкции или из нескольких инструкций, заключенных в фигурные скобки.

Цикл while

Синтаксис цикла while:



Инструкция while сначала вычисляет значение выражения. Полученное значение, если необходимо, неявно преобразуется к логическому типу. само выражение является условием цикла. Если выражение имеет ложное значение, интерпретатор не выполняет тело цикла, и переходит к выполнению следующей инструкции в коде программы. Если результатом вычисления выражения является истинное значение, то тело цикла выполняется, затем управление передаётся в начало цикла и выражение вычисляется снова. Иными словами, интерпретатор снова и снова выполняет инструкции расположенные в теле цикла, пока значение выражения остаётся истинным.

Рассмотрим простой пример цикла while:

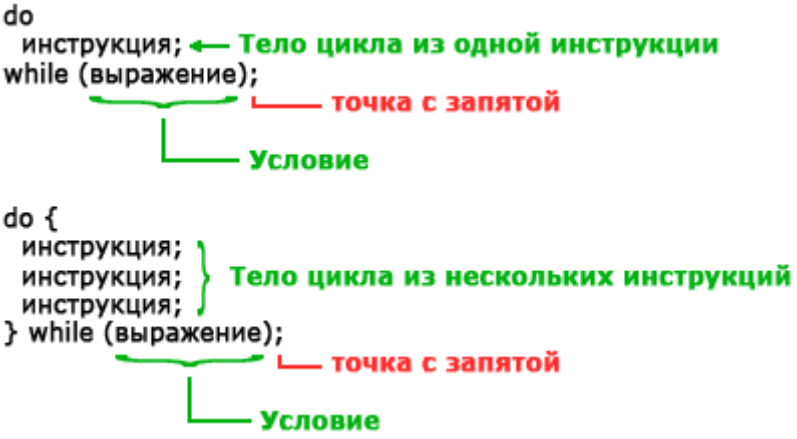
```
var count = 0; // определяем переменную счётчик
while (count < 3) {
  document.write(count + " ");
  count++;     // если из кода убрать эту строку, то цикл будет бесконечным
}
```

Большинство циклов имеют переменные-счётчики, аналогичные переменной count. Чаще всего в качестве счётчиков цикла выступают переменные с именами i, j и k, хотя для того чтобы сделать программный код более понятным, следует давать счётчикам более наглядные имена.Перед началом цикла значение переменной count устанавливается равным 0 (значение может быть любым). Это называется инициализацией переменной-счётчика. Значение выражения (count < 5) проверяется каждый раз перед итерацией цикла. Когда исполняется тело цикла, с помощью инкремента count++ значение переменной count увеличивается на единицу. После пяти итераций выражение вернёт значение false (так как значение переменной count уже будет не меньше 5) и работа цикла прекратится.

```
var i = 3;
while (i) { // при i, равном 0, значение в скобках будет false и цикл остановится
  alert( i );
  i--;
}
```

Цикл do/while

Синтаксис цикла do/while:



Цикл do/while похож на цикл while, за исключением того, что проверка условия выполнения цикла производится после итерации, а не в начале, и завершается цикл точкой с запятой. Так как условие проверяется в конце, тело цикла do/while всегда выполняется минимум один раз:

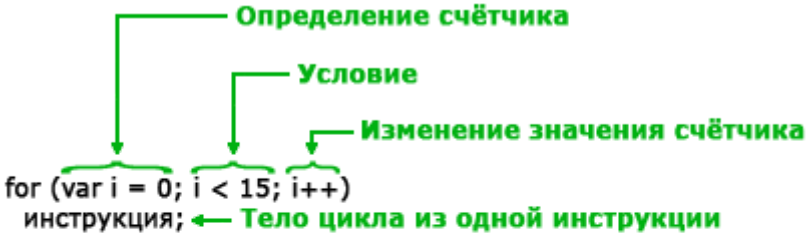
```
var count = 0;
do {
  count++;
  alert(count);
} while(count < 5);
```

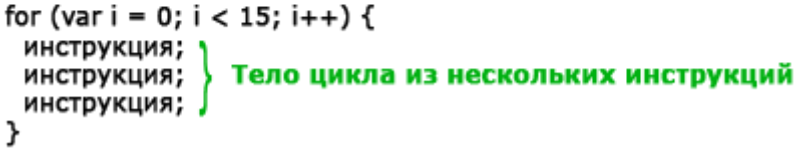
Данный цикл может быть полезен, когда код в теле цикла должен быть выполнен хотя бы один раз, независимо от условия выполнения.

Цикл for

Большинство циклов имеют некоторую переменную-счётчик. Эта переменная инициализируется перед началом цикла и проверяется перед каждой итерацией. Далее переменная-счётчик изменяется каким-либо образом в конце тела цикла, непосредственно перед повторной проверкой условия выполнения цикла.

Инициализация, проверка и обновление - три ключевых операции, выполняемых с переменной цикла. Эти три шага являются явной частью синтаксиса цикла for:





Таким образом, все элементы, контролирующие выполнение цикла for, собраны в одном месте, в то время как в циклах других типов они находятся в различных частях кода.

Цикл for состоит из ключевого слова for, за которым следуют круглые скобки, внутри которых располагаются три выражения, разделяемые точками с запятой. Он имеет следующий порядок выполнения:

1.Первое выражение всегда вычисляется только один раз - в начале цикла. В цикле for допускается определять переменные, поэтому обычно в качестве первого выражения выступает **определение переменной-счётчика**.

2.Второе выражение определяет условие выполнения цикла. Оно вычисляется перед каждой итерацией и определяет, будет ли выполняться тело цикла. Если результатом вычисления выражения является истинное значение, инструкции, являющиеся телом цикла - выполняются. Если возвращается ложное значение, выполнение цикла завершается. Если при первой проверке условия, оно оказывается ложным, тело цикла не выполнится ни разу.

3.После каждой итерации вычисляется третье выражение. Обычно его используют для изменения значения переменной-счётчика, которая используется в проверке условия выполнения цикла.

Небольшой пример, в котором выводятся цифры от 0 до 3:

```
for (var count = 0; count < 4; count++)
  alert(count + " ");
```

Любое из выражений может отсутствовать, однако сами точки с запятой (;) обязательно должны присутствовать, иначе будет ошибка синтаксиса. Если второе выражение отсутствует, это означает, что цикл будет выполняться бесконечно.

```
var i = 0;
for (; i < 4; i++) ...

var i = 0;
for (; i < 4; ) ...

for (var i = 1; /* нет условия */ ; i++) ...
```

Это эквивалентно следующему коду

```
for (var i = 1; true; i++) ...
```

Вместо одного выражения можно указать несколько выражений, разделяя их оператором запятая.

Не выполнится, так как в проверке условия последнее выражение false

```
for (i = 1; i < 4, false; i++) ...
for (var i = 1, j = 5; i <= 5; i++, j--)
  document.write(i + " " + j +"<br>");
```

Вложенные циклы

Циклы могут иметь любое количество уровней вложенности:

```
for(var i = 0; i < 3; i++) {
  alert("Часть внешнего цикла. <br>");
  for(var j = 0; j < 2; j++) {
    alert("Часть вложенного цикла. <br>");
  }
}
var i = j = 0;
while (i < 3) {
  j = 0;
  document.write("Часть внешнего цикла. <br>");
  while(j < 1) {
    document.write("Часть вложенного цикла. <br>");
    j++;
  }
  i++;
}
```

Инструкции break и continue

Иногда бывают ситуации, когда необходимо прервать выполнение цикла. Инструкция break приводит к немедленному выходу из текущего цикла. Синтаксис инструкции достаточно прост:

```
break;
```

Ниже показано применение инструкции break:

```
for(var i = -10; i <= 10; i++) {
  if(i > 0) break;
     alert(i + " ");
}
alert("Готово!");
var j = -1;
for(var i = 0; i < 3; i++) {
 document.write("Часть внешнего цикла. <br>");
 while (j < 5) {
  j++;
  if (j == 2 || j == 3) break;
  document.write("j: "+ j +"<br>");
 }
}
```

Инструкция **continue** схожа с инструкцией **break**, однако вместо выхода из цикла она запускает новую итерацию цикла. Инструкция continue может использоваться только в циклах. Синтаксис инструкции continue также прост, как и синтаксис инструкции break:

```
continue;
```

Когда выполняется инструкция **continue**, текущая итерация цикла прерывается и начинается следующая. Для разных циклов инструкция даёт разный эффект:Инструкция **continue** схожа с инструкцией **break**, однако вместо выхода из цикла она запускает новую итерацию цикла. Инструкция continue может использоваться только в циклах. Синтаксис инструкции continue также прост, как и синтаксис инструкции break:

- После выполнения инструкции **continue** в цикле **while** проверяется условие выполнения, и если оно имеет значение true, тело цикла выполняется.
- В цикле for после выполнения инструкции continue **сначала** вычисляется третье выражение, и только затем происходит проверка условия.
- В цикле do/while после выполнения инструкции **continue** происходит переход в конец цикла и проверяется условие выполнения, если оно имеет значение true, тело цикла выполняется.

Ниже приведён пример, в котором инструкция continue используется в качестве вспомогательного средства для вывода чётных чисел в пределах от 0 до 10:

```
for (var i = 0; i <= 10; i++) {
 if((i % 2) != 0) continue;
     alert(i);
}
```

2.4 Инструкция switch в JavaScript

Инструкция switch используется для выбора одного из нескольких вариантов действий в зависимости от результата сравнения значения выражения с несколькими константами. Синтаксис инструкции switch выглядит следующим образом:

```
switch (выражение) {
 case константа: инструкции;
 ...
```

```
  case константа: инструкции;
  default: инструкции;
}
```

Инструкция switch сравнивает значение выражения, расположенного в круглых скобках, со всеми константами, стоящими рядом с ключевыми словами case, в порядке их следования. При сравнении используется оператор идентичности "===". Каждый из вариантов (блоков case) имеет метку в виде константного значения, за которым ставится двоеточие. Если одна из меток совпадает со значением выражения, то управление передаётся инструкциям, расположенным после этой метки. Блок default выполняется в том случае, если не найдено ни одного соответствия с метками блоков case. Наличие блока default не обязательно, если его нет и не найдено ни одного соответствия, то никакие инструкции выполнены не будут:

```
var x = 1;
switch (x) {
  case 1:
    alert("x равен 1");
    break;
  case 2:
    alert("x равен 2");
    break;
  case 3:
    alert("x равен 3");
    break;
  default:
    alert("x > 3");
}
```

Инструкция break инициирует немедленный выход из инструкции switch, и далее управление передаётся инструкции, следующей за конструкцией switch.

При отсутствии инструкции break управление будет передано инструкциям, находящимся в других блоках case, т.е. после выполнения кода в блоке case продолжается выполнение кода следующего блока case, при этом остальные проверки игнорируются. Пример без инструкции break:

```
var x = 2;
switch (x) {
  case 1:
    document.write("x равен 1");
  case 2:
    document.write("x равен 2");
  case 3:
    document.write("x равен 3");
  default:
    document.write("x > 3");
}
```

Для выхода из **switch** может использоваться не только инструкция break, но также и инструкция return (если switch находится внутри функции) или continue (если switch находится внутри цикла).

Достоинство в отсутствии инструкции break состоит в том, что несколько возможных вариантов case можно ассоциировать с одним и тем же набором выполняемых инструкций.Для выхода из **switch** может использоваться не только инструкция break, но также и инструкция return (если switch находится внутри функции) или continue (если switch находится внутри цикла).

```
var x = 3;
```

```
switch (x) {
  case 1:
  case 2:
  case 3:
    document.write("x равен 1, 2 или 3");
    break;
  case 7:
    document.write("x равен 7");
    break;
}
```

Блоки case не обязательно должны записываться друг под другом, для удобства их можно расположить в ряд:

```
var x = 3;
switch (x) {
  case 1: case 2: case 3:
    document.write("x равен 1, 2 или 3");
    break;
  case 7:
    document.write("x равен 7");
    break;
}
```

Объект Math

Объект Math используется для выполнения математических операций и извлечения значений констант.

Синтаксис:

Math.константа

Math.функция()

Тригонометрические методы	
cos(x)	косинус числа x
sin(x)	синус числа x
tan(x)	тангенс числа x
acos(x)	арккосинус числа x
asin(x)	арксинус числа x
atan(x)	арктангенс числа x
Методы вычислений	
exp(x)	e^x
log(x)	натуральный логарифм ln(x)
pow(x, y)	x^y
sqrt(x)	квадратный корень от x
Константы	
E	Число e – основание натурального логарифма (≈ 2.72)
LN10	Число ln(10) ≈ 2.3
LN2	Число ln(2) ≈ 0.69
LOG10E	Число lg(e) ≈ 0.43
LOG2E	Число $\log_2(e)$ ≈ 1.44
PI	Число π ≈ 3.14
SQRT1_2	Квадратный корень от ½ ≈ 0.71