Функции

Функция представляет собой подпрограмму, которую можно вызвать для выполнения, обратившись к ней по имени.

Взаимодействие функции с внешней программой, из которой она была вызвана, происходит путем передачи функции параметров и приема от нее результата вычислений. Функция в JavaScript может и не требовать параметров, а также ничего не возвращать.

В JavaScript есть встроенные функции, которые можно использовать в программах, но код которых нельзя редактировать или посмотреть. Все, что мы можем узнать о них, — это описание их действия, параметров и возвращаемого значения.

Кроме использования встроенных функций вы можете создать свои собственные, так называемые пользовательские функции.

Часто используемые фрагменты программного кода целесообразно оформлять в виде функций. Такой фрагмент кода заключается в фигурные скобки, а перед ним пишется ключевое слово **function**, за которым следуют круглые скобки, обрамляющие список параметров.

Чтобы вызвать функцию в программе, следует написать выражение в следующем формате:

имя_функции (параметры)

Если требуются параметры, то они указываются в круглых скобках через запятую. Функция может и не иметь параметров. В этом случае в круглых скобках ничего не указывается.

Встроенные функции

Рассмотрим некоторые встроенные функции:

eval(строка) — вычисляет выражение в указанной строке; выражение должно быть написано на языке Java Script (не содержит тегов HTML).

Пример

```
var y = 5                                   // значение y равно 5
var x = "if(y<10) {y = y+2}"                // значение x равно строке символов
eval(x)                                     // значение y равно 7
```

Приведем текст соответствующего HTML-кода со сценарием, содержащим функцию **eval():**

```
<html> <body>
<textarea  id = "mycode”   rows=10  cols = 60>          </ textarea >
< textarea id = "myrezult"   rows = 3 cols = 60>        </ textarea >
<p>  <button onclick =
    "document.all.myrezult.value=eval(mycode.value)"> Выполнить </
    button >
< button onclick="document.all.mycode.value=";
    document.all.myrezult.value="">
Очистить</ button >
</p> </body></html>
```

- **Функции в JavaScript** являются объектами, и как следствие могут, например, присваиваться переменным, передаваться другим функциям, можно присваивать значения их свойствам и вызывать их методы.
- Основное назначение функции заключается в том, чтобы избавить программу от дублирования кода. Записав определенную последовательность команд можно вызывать ее сколько угодно раз.

Пример:

```
<html>
<body>
<script>
document.write(escape("How do you do")) ;
document.write(unescape("How%20do%20you%20do"));
</script>
</body>
</html>
```

typeof() — возвращает тип указанного объекта в виде символьной строки;

например "boolean", "function" и т. п.

Пользовательские функции

Функция задается своим определением (описанием), которое начинается ключевым словом function. Синтаксис:

function имя_функции (параметры)

{ код }

Часто определение функции записывают и в таких формах:

Имя функции выбирается по тем же правилам, как и имя переменной. Недопустимо использовать в качестве имени ключевые слова языка JavaScript.

За именем функции обязательно стоит пара круглых скобок. Программный код (тело) функции заключается в фигурные скобки. Они определяют группу выражений, которые относятся к коду именно этой функции.

Если функция принимает параметры, то список их имен (идентификаторов) указывается в круглых скобках около имени функции. Если параметров несколько, то в списке они разделяются запя тыми. Если параметры для данной функции не предусмотрены, то в круглых скобках около имени функции ничего не пишут.

Когда создается определение функции, список ее параметров (если он необходим) содержит просто формальные идентификаторы (имена) этих параметров, понимаемые как переменные.

В определении функции в списке параметров, заключенном в круглые скобки сразу же за именем функции после ключевого слова **function**, ***нельзя использовать конкретные значения и выражения***. В этом смысле определение функции задает код, оперирующий формальными параметрами, которые конкретизируются лишь при вызове функции из внешней программы.

Если требуется, чтобы функция возвращала некоторое значение, то в ее теле используется оператор возврата **return** с указанием справа от него того, что следует возвратить. В качестве возвращаемой величины может выступать любое выражение: простое значение, имя переменной или вычисляемое выражение.
Пример : return x;

Оператор **return** может встречаться в коде функции несколько раз.

```
function divideOneTo(x)
{   if (x != 0) {     return 1/x;   }
            else {   return "А на ноль делить нельзя!";   } }
```

Использование ключевого слова **return** не является обязательным. Если в теле функции его нет, то после выполнения всех команд интерпретатор закончит выполнение функции. Значение, которое возвращает эта функция будет "**undefined**".

Пример

Допустим, требуется определить функцию для вычисления площади прямоугольника. Назовем эту функцию Srect. Тогда ее определение будет выглядеть следующим образом:

```
function Srect(width, height){
S = width * height;
return S;  }
```

Здесь сначала вычисляется площадь путем умножения значений параметров width и height, полученное значение присваивается переменной S, а затем оператор return возвращает значение этой переменной.

Определение функции Srect можно сделать более экономным, минуя присвоение значения переменной S:

```
function Srect(width, height){
return width * height
}
```

Определение функции, описанное выше, будучи помещенным в программу, усваивается интерпретатором, но сама функция (ее код или тело) не выполняется.

Чтобы выполнить функцию, определение которой задано, необходимо написать в программе выражение вызова этой функции. Оно имеет следующий синтаксис:

имя_функции (параметры)

Имя функции должно полностью (вплоть до регистра) совпадать с именем ранее определенной функции. Параметры, если они заданы в определении функции, в вызове функции представляются конкретными значениями, Переменными или выражениями.

Не путайте определение функции с ее вызовом!

В JavaScript можно не поддерживать равенство между количествами параметров в определении функции и в ее вызове.

Если в функции определены, например, три параметра, а в вызове указаны только два, то последнему параметру будет автоматически присвоено значение **null**. Наоборот, лишние параметры в вызове функции будут просто проигнорированы.

Примеры: вызов функции

Srect(3, 5) /* возвращает площадь прямоугольника размером 3x5 */

Srect (3, 4 + 2) /* возвращает площадь прямоугольника размером 3x6 */

height = 8
Srect (3, height) /* возвращает площадь прямоугольника размером 3x8 */

width=4
height=5
Srect (width, height + 2) /* возвращает площадь прямоугольника размером 4x7 */

Srect (2) /* возвращает 0 (второй параметр не задан) */

Переменные в JavaScript могут быть Локальными и Глобальными

- **Локальные переменные -** это переменные, объявленные внутри функции JavaScript. Они доступны только в пределах той функции, внутри которой они объявлены. При выходе из этой функции переменные уничтожаются.
- Можно объявлять внутри разных функций переменные с одинаковым именем - они никак не будут пересекаться, поскольку используются только внутри функции, в которой они созданы.
- **Глобальные переменные** объявляются вне функций и к ним могут обращаться все функции и скрипты на странице. Уничтожаются такие переменные при закрытии страницы.
- Если переменную объявить без использования ключевого слова "var", то она автоматически объявляется глобальной, даже если объявление произведено внутри функции.
- Например выражения **x = 5;** или **surName = "Ivanov";** объявят переменные **x** и **surName** как глобальные, если их еще не существует.

Область видимости переменных

Если вы используете просто **оператор присвоения**, то возможны две ситуации.

1. Переменная в операторе присвоения встречается первый раз в вашей программе именно **в теле функции**. В этом случае она действует в пределах кода (тела) этой функции, но после выполнения функции эта переменная **продолжает существовать и во внешней программе**. Таким образом, эта переменная является ***глобальной***.
2. Переменная в операторе присвоения уже определена во внешней программе.

В этом случае она и действует внутри функции, и продолжает существовать после завершения ее выполнения, поскольку она была создана во внешней программе как глобальная переменная.

Если в теле функции вы используете ключевое слово **var** для инициализации переменной, то эта переменная будет действовать только в пределах кода функции, независимо от того, была ли она определена во внешней программе или нет. При этом если переменная создана с ключевым словом **var** впервые в теле функции, то она является **локальной**, то есть недоступной из внешней программы. Таким образом, инициализация переменной с помощью выражения с ключевым словом **var** создает **локальную переменную**.

Если в теле функции вы используете переменную, определенную только во внешней программе, то изменение ее значения в теле функции сохранится даже после завершения выполнения этой функции, поскольку это глобальная переменная.

Если во внешней программе вы определили некоторые переменные, имена которых совпадают с формальными параметрами в определении функции, а затем указываете их в качестве параметров вызова этой функции, то произойдет следующее. Функция воспримет значения параметров, определенные во внешней программе, однако любые их изменения в теле функции останутся локальными, то есть после завершения работы кода функции значения указанных переменных останутся прежними, как до вызова функции.

Программа может содержать и определение функции, и выражения ее вызова. При этом порядок их следования в программе не важен: вы можете сначала написать определение функции, а затем где-нибудь в программе написать ее вызов, или, наоборот, написать сначала вызов функции, а ее определение разместить где-нибудь в конце программы или даже в отдельном файле с расширением .js.

Однако принято описывать все функции (то есть код, который может сработать во время работы программы, а может и остаться невостребованным) в области **head** документа для того, чтобы не запутывать код выполнения программы.

При определении функции после ключевого слова может не быть имени

Например, можем объявить функцию вот таким способом:

var printText = function(a)

{document.write(a);};

Функция может быть сразу вызвана с необходимым входным параметром. В следующем примере функция сразу будет вызвана - выведет в документ фразу "Hello World!". Если бы функция возвращала какой-либо результат, он бы был записан в переменную printText.

var printText = function(a)

{document.write(a);}("Hello World!");

- Для того чтобы обозначить что функция является самовызываемой, ее заключают в скобки. Это не требование стандарта, однако так делают для большей очевидности.
- var printText = (function(a) {document.write(a);}("Hello World!"));

Самовызывающиеся функции в JavaScript

Синтаксис:

(function () { // код функции }());

Главной идеей является то, что анонимная функция вызывается сразу после своего объявления. Преимущество от использования самовызывающихся функций вы получите, если нужно выполнить код один раз и сохранить его результаты во "внешней среде" (без объявления глобальных переменных).

- Например, для небольшой веб-страницы можно написать обработчики событий для элементов на странице. Самовызывающиеся функции подходящее средство для такой ситуации.

Пример вычисления факториала числа

```
<html>     <head>          <script language="JavaScript">
                           function factorial(n){
                           if(n<=1) {return 1};
                           result = 2 ;
                           for (i=3;i<=n;i++) {
                           result=result*i;      }
                           return result;          }
                           </script>
           </head>
 <body>
<script language="JavaScript">
     var m=10;
     x=factorial(m)
     document.write("факториал от числа"+m+"   равен   "+x);
</script>  </body>  </html>
```

Рекурсия

Определение функции может содержать вызов этой же функции — так называемое рекурсивное определение функции.

Пример: Функция вычисления факториала n! с помощью рекурсии.

```
<html>   <head>  <script>
          function factorial(n){
            if(n <= 1){ return 1 }
            return n*factorial(n-1) } // вызов функции factorial()
</script> </head> <body>
                  <script>
   document.write(factorial(9));</script>
</body> </html>
```

СОВЕТ

Чтобы избежать ошибок, старайтесь не использовать в коде функций имена переменных, инициализированных во внешней программе. По возможности стремитесь к тому, чтобы все, что делается внутри вашей функции, было локальным, формально независимым от внешнего окружения.

Важное замечание: в рекурсивных функциях нужно не забывать про условие прекращения рекурсии, иначе код "зациклится".

В примере выше, таким условием будет проверка, что n <= 1