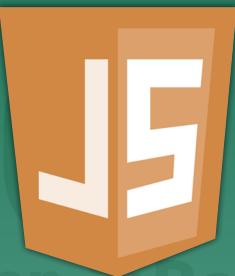


Разработка клиентских сценариев с использованием JavaScript и библиотеки jQuery



Unit 4

Функции

Contents

Что такое функция?	3
Синтаксис объявления функции.....	7
Параметры функции.....	11
Возвращаемое значение функции.	
Ключевое слово return.....	18
Задание для самостоятельной работы	23

Материалы урока прикреплены к данному PDF-файлу. Для доступа к материалам, урок необходимо открыть в программе Adobe Acrobat Reader.

Что такое функция?

Рассмотренные в предыдущем разделе циклы позволяют многократно исполнять один и тот же блок программы, но только все его повторы будут происходить последовательно — один за другим. После прекращения работы цикла возврат в его тело невозможен (если он, конечно, не вложен в другой цикл). Однако в реальных программах часто появляется необходимость повторять один и тот же блок, но в разных ее участках, а не только последовательно.

Приведем аналогию, напрямую не связанную с программированием. Представим, что мы сотрудники канцелярии и наша задача состоит в подготовке ответов на обращения клиентов некоторой организации, то есть мы должны составлять и отправлять письма с ответами на поступающие вопросы. Так как организация из года в год ведет одну и ту же деятельность, вопросы клиентов будут довольно часто повторяться или, по крайней мере, будут очень похожими на какие-то из предыдущих вопросов. Набравшись определенного опыта мы, наверняка, составим набор заготовок писем, в которые надо будет подставить только конкретные имена, адреса, даты и всё — письмо готово к отправке. Наличие таких шаблонов-заготовок значительно упростит нам работу, исключит потенциальные ошибки, ускорит рабочий процесс.

При составлении компьютерных программ возникают подобные ситуации. Определенный набор действий приходится повторять в разных частях одной программы

или в нескольких разных программах. При этом в ряде случаев в действия нужно «подставить» новые данные, а иногда и полностью повторить весь код. Конечно, есть возможность обычного текстового клонирования (копирования-вставки), но это увеличивает размер кода, ухудшает его читаемость и может приводить к проблемам повторного использования одних и тех же переменных, т.к. при копировании будут повторены все описанные имена. Хотелось бы иметь возможность создания неких шаблонов-заготовок кода, которые можно было бы использовать, ссылаясь на них, а не копируя содержимое.

Для того чтобы реализовать такую возможность был разработан механизм функций. В программировании функциями называют отдельные самостоятельные фрагменты кода, которые могут быть вызваны в разных местах основной программы (или в других функциях) при помощи ссылки на их имя. Единожды составив и настроив код, выполняющий некоторое действие (подготовив шаблон), мы можем оформить его в виде функции и свободно использовать в произвольных местах, в которых нужно воспользоваться этим кодом.

На самом деле мы уже неоднократно сталкивались с использованием функций. Например, диалоговые окна (`alert`, `prompt` и `confirm`) являются типичными их представителями. Рассмотрим, что происходит, когда в программе появляется инструкция «`alert(x)`».

Интерпретатор языка JavaScript обнаруживает имя «`alert`» и устанавливает, что за ним закреплена определенная функция (подпрограмма) и в эту функцию нужно передать значение переменной «`x`». То есть для работы

функции нужно подставить определенные данные вместо «*x*». Дальше интерпретатор останавливает выполнение основной программы и переходит к подпрограмме, «запомнив» значение «*x*». Это называется вызовом функции (*function call*).

Инструкции функции «`alert`» рисуют окно и выводят в него запомненное при вызове значение «*x*» (подставляют в шаблон конкретные данные). После чего ожидают закрытия окна пользователем. Подпрограмма заканчивается, и интерпретатор возвращается к тому месту, откуда произошел ее вызов. Этот процесс называется возврат из функции (*function return*).

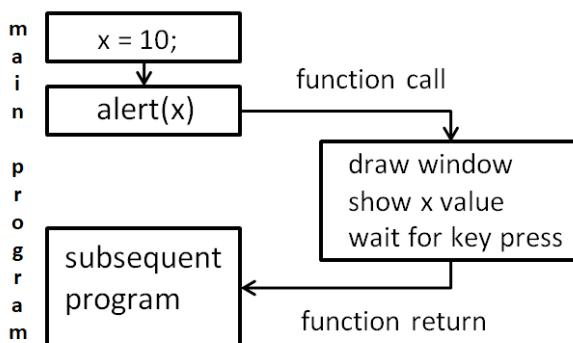


Рисунок 1

При возврате интерпретатор также может «запомнить» некоторые данные, которые следует передать из функции в основную программу. Например, когда нам был нужен пользовательский ввод, мы использовали инструкцию

```
x = prompt("Input x=")
```

Оператор присваивания сначала должен вычислить выражение в правой части от знака «`=`». При этом происходит вызов функции: интерпретатор запоминает то, что надо для работы функции (строка `«Input x=»`), и передает управление в подпрограмму с именем `«prompt»`. После окончания работы функции результат, полученный от пользователя в окне, будет запомнен интерпретатором и перемещен в переменную `«x»` при возврате в основную программу.

В виде функций реализуются дополнительные возможности программирования, прямо не предоставляемые набором базовых операторов языка. Многие из таких функций изначально включены в состав JavaScript, например, диалоговые окна или математические функции. Для того чтобы разделять функции, поставляемые с языком, и функции, созданные программистом в своей программе, последние также называют «пользовательскими» функциями. Принципиальной разницы между стандартными и пользовательскими функциями нет, название лишь уточняет, о каком виде функций идет речь.

Забегая наперед отметим, что вызов функции не всегда останавливает выполнение основной программы. Есть особый вид функций, которые могут выполняться параллельно с работой программы и других функций. Они носят название «асинхронных». При этом функции, которые останавливают работу программы, называют «синхронными». Пока что мы будем иметь в виду только синхронные функции, об асинхронных будет рассказано в дальнейших разделах.

Синтаксис объявления функции

Пользовательские функции создаются при помощи ключевого слова «**function**». Синтаксис объявления функции имеет следующий вид:

```
function nameOfFunction(argument1, argument2) {  
    body  
}
```

После ключевого слова «**function**» следует имя функции (**nameOfFunction**), созданное согласно правилам (и рекомендациям) именования. После имени в круглых скобках указываются параметры, которые будут приниматься в функции. Если параметры не нужны, то круглые скобки оставляют пустыми (но не убирают). Количество параметров принципиально не ограничено.

```
function functionWithoutArguments () {  
    body  
}
```

Далее в фигурных скобках описывается тело функции, то есть сам код функции как самостоятельной программы или шаблона для подстановки в другие места программы. В отличие от условных и цикловых операторов, в случае с объявлением функции фигурные скобки обязательны, даже если функция состоит из одного оператора.

Как уже было сказано, тело функции можно представить себе, как отдельную программу. Эта программа является достаточно независимой: в ней могут описываться собственные переменные, организовывать свои циклы, проверяться необходимые условия и т.п. — в теле функции доступны все средства программирования.

Рассмотрим в качестве примера функцию,ирующую 5 однотипных блоков-заголовков (`<h2></h2>`) с надписями `Header 1 ... Header 5`. Создайте новый файл, наберите или скопируйте в него следующее содержание (код также доступен в папке Sources — файл `js1_7.html`).

```
<!doctype html>
<html>
    <head>
    </head>

    <body>
        <script>
            function show5Blocks() {
                for(i=1;i<=5;i++)
                    document.write("<h2> Header "+i+
                                "</h2>");

            }
            show5Blocks();
        </script>
    </body>

</html>
```

Изначально, страница, создаваемая данным файлом, пустая. В ней нет никаких HTML элементов. В блоке сце-

нариев (`<script>`) описывается функция `show5Blocks()`. В теле функции используется цикл-счетчик «`for`», который 5 раз записывает на страницу (командой `document.write`) строку, сформированную разметкой «"`<h2> Header +i+"</h2>"`». В данном случае оператор «`+`» используется для сложения строк. В результате будут получаться требуемые надписи `Header 1 ... Header 5` для каждого значения `i`, перебираемого циклом.

Обратим внимание, что само по себе описание функции не выполняет действий, указанных в ее теле. Для того чтобы эти действия запустились, функцию необходимо вызвать. Это обеспечивается указанием ее имени, после которого ставятся круглые скобки «`show5Blocks()`». Именно круглые скобки сообщают интерпретатору про необходимость вызова функции, то есть про переход к командам ее тела.

Сохраните файл и откройте его в браузере. В результате на странице должны появиться 5 заголовков:

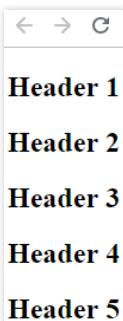


Рисунок 2

Приведенный пример, кроме определения функции, дополнитель но демонстрирует их возможности по соз-

данию новых элементов Веб-страницы. После работы функции на изначально пустой странице появляются HTML заголовки. Их содержание можно просмотреть при помощи средств разработчика и убедиться, что они являются полноценными элементами страницы, имеют все необходимые HTML атрибуты и свойства.

Параметры функции

В силу определенной независимости функции в ней может быть ограничен доступ к переменным из основной программы или других функций, откуда происходит ее вызов. Сравнивая функцию с некоторым шаблоном-заготовкой письма, можно привести аналогию, что при со-ставлении шаблона мы не можем знать конкретных дан-ных для записи, а можем лишь указать, что здесь нужно указать конкретный адрес, а здесь — адресата. Конкрет-ные данные будут известны, когда мы будем использовать шаблон, то есть в аналогии — вызывать функцию.

Если в теле функции нужно использовать какие-либо данные для подстановки извне, то необходимо принять меры по обеспечению доступа к ним. В компьютерной терминологии это называется механизмом «передачи данных в функцию». Передаваемые из программы данные называются «аргументами», а со стороны функции эти данные принимаются как «параметры». В используемой нами аналогии, параметры — это пока что пустые места, куда будут вписаны адрес или адресат, а аргументы — это сами значения адреса и адресата.

Технически, параметры функций играют роль пере-менных, значения которых устанавливаются в момент вызова функции другой программой. В теле функции параметры можно использовать в тех же языковых ин-струкциях, что и обычные переменные. При вызове функ-ции вместо параметров будут подставлены конкретные значения аргументов.

Рассмотрим пример, иллюстрирующий передачу данных: необходимо создать функцию, которая будет увеличивать значение аргумента на 1 и выводить полученный результат на страницу.

Этап 1: подготовка окружения. Создайте новый файл, наберите или скопируйте в него следующее содержание (код также доступен в папке Sources — файл js1_8.html)

```
<!doctype html>
<html>
  <head>
  </head>
  <body>
    <p id="Log"></p>
    <script>
    </script>
  </body>
</html>
```

Документ содержит основные разметочные определения HTML страницы, в состав которой входит абзац с идентификатором «Log» (`<p id="Log"></p>`), предназначенный для будущего отображения данных.

Этап 2: определение функции. В раздел `<script>` помещаем следующий код

```
function incAndLog(x) {
  x = x+1;
  alert("inc x = " + x);
  Log.innerHTML += "<br>inc x = " + x;
}
```

Для имени функции выбран литерал «`incAndLog`», символизирующий смысл работы функции: инкремент (увеличение на 1) и вывод результата в лог (абзац с идентификатором «`Log`»). Придерживаемся правил написания составных имен «`lowerCamelCase`».

В качестве параметра функции после ее имени в круглых скобках указываем «`x`». На данном этапе параметр называется формальным, то есть он еще не имеет значения, но может использоваться в теле функции как обычная переменная. В приведенной выше аналогии с письмами формальный параметр — это пустое место шаблона для записи конкретных данных (адреса).

Согласно поставленной задаче, увеличиваем параметр на 1 (`x = x + 1`) и выводим полученное значение, формируя сообщение «`"inc x = " + x`» (надпись «`inc x =`» и плюс само значение переменной «`x`»).

Вывод сообщения обеспечиваем двумя способами. Во-первых, вызываем диалоговое окно «`alert`» с указанным текстом сообщения. Во-вторых, добавляем полученное сообщение в состав абзаца (`Log.innerHTML`), дополнив сообщение разрывом строки «`
`». Напомним, что обращение к элементу с заданным идентификатором возможно по его имени (`Log`).

Этап 3: вызов функции и передача аргументов. После определения функции в раздел `<script>` добавим следующий код:

```
x = 2;  
incAndLog(x);
```

Сначала вводится переменная «`x`» и ей присваивается значение 2. Затем вызывается функция `incAndLog(x)`, в которую передается аргумент «`x`».

При обращении к созданному файлу браузер начнет загружать страницу. Начиная от начала разметки, первым будет создан абзац (`<p id="Log"></p>`). Затем начнется выполнение блока сценария (скрипта). В нем сначала будет определена функция «`incAndLog`». Само определение функции не приводит к выполнению ее тела, только «заготовливает» ее для будущего использования.

После определения, вводится переменная «`x=2`» и вызывается функция «`incAndLog(x)`». Во время вызова формальный параметр «`x`» в теле функции связывается с аргументом «`x`», определенным как переменная перед вызовом функции. То есть параметр приобретает значение и в дальнейшем теле функции в качестве начального значения параметра «`x`» будет использовано значение «`2`», переданное при вызове функции.

В теле функции значение «`x`» увеличивается на 1 (`x=x+1`), после чего формируется сообщение с учетом нового значения параметра (`2+1=3`). Сообщение выводится при помощи диалогового окна и, затем, это же сообщение копируется на страницу.

Описанную последовательность вызовов функций при работе созданного сценария можно представить в виде схемы (см. рис. 3).

Сохраните созданный файл и откройте его при помощи браузера. После загрузки страницы должно появиться диалоговое окно с сообщением «`inc x = 3`». Обратите внимание, что сама страница остается пустой. То есть

сообщение не дублируется на странице, пока выполняется функция диалогового окна.

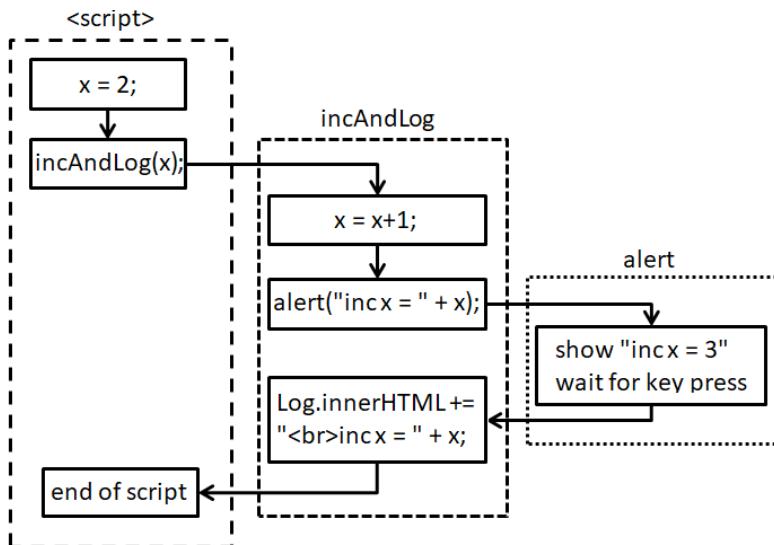


Рисунок 3

Закройте диалоговое окно и убедитесь, что на странице появляется такое же сообщение «`inc x = 3`».

Следует обратить внимание, что при передаче аргумента в функцию его значение попадает в переменную-параметр `«x»` путем копирования. В результате чего изменения, внесенные в значение параметра (`$x = x + 1$`) не отображаются на значениях аргумента в точке вызова функции, т.к. функция работает с копией аргумента. Проверим это на следующем примере.

Внесите изменения в предыдущий или создайте новый файл, наберите или скопируйте в него следующее содержание (код также доступен в папке Sources — файл `js1_9.html`).

```

<!doctype html>
<html>
    <head>
    </head>
    <body>
        <p id="Log"></p>
        <script>
            function incAndLog(x) {
                x = x+1;
                alert("inc x = " + x);
                Log.innerHTML += "<br>inc x = " + x;
            }
            x = 2;
            Log.innerHTML = "x = " + x;
            incAndLog(x);
            Log.innerHTML = "<br>x = " + x;
        </script>
    </body>
</html>

```

Дополнительно к предыдущему примеру на страницу выводятся сообщения о значении переменной «`x`» до и после вызова функции. Вполне очевидно, что до вызова функции значение «`x`» равно `2`. А вот что произойдет после выполнения тела функции, ведь в ней используется переменная-параметр с точно таким же именем и она увеличивается на `1`?

Сохраните файл и откройте его при помощи браузера. Убедитесь, что окно «`alert`» содержит такое же сообщение, как и в предыдущем примере (`inc x = 3`). После закрытия окна на странице появляются три строки сообщения.

Первая строка выводится до вызова функции и содержит, как и ожидалось, значение `2`. Затем идет стро-

ка, сформированная самой функцией, сообщающей, что внутри функции «`x`» имеет значение `3`. Третей идет строка, отображающая значение «`x`» после вызова функции. Как видно (см. рис. 4), это значение все так же равно `2`. Убеждаемся в том, что функция работала со своей копией аргумента и не повлияла на внешнюю переменную даже при полном совпадении их имен.

```
← → C
x = 2
inc x = 3
x = 2
```

Рисунок 4

Возвращаемое значение функции. Ключевое слово return

Для обеспечения обратного потока данных, — из функции в основную программу, — применяется механизм возврата значений. Подобно тому, как значения из программы попадают в функцию через параметры, значения из функции могут попасть в программу (в точку вызова функции).

В объявлении функции для возврата значения используется ключевое слово «`return`», после которого указываются передаваемые из функции данные. Выполнение команды `«return»` прекращает работу функции, даже если в ее теле есть дальнейшие инструкции. Управление передается в точку вызова функции, а переданные данные могут быть использованы как результат, заменяющий собой имя функции. Этот результат может быть помещен (присвоен) в некоторую переменную, либо включен в состав выражения.

Рассмотрим пример: необходимо создать функцию, вычисляющую куб переданного аргумента (напомним, куб — это результат умножения числа самого на себя 3 раза: $x^3=x\cdot x\cdot x$).

Этап 1: определение функции

```
function cube(x) {
    return x*x*x;
}
```

Расчет значения куба числа достаточно несложный, поэтому нет необходимости создавать вспомогательные переменные или использовать дополнительные языковые конструкции. Тело функции состоит из единственного выражения «`return x*x*x;`». Его выполнение заключается в вычислении умножения `«x*x*x»` и запуска механизма возврата полученного результата.

Для исследования таких небольших функций удобно использовать консоль браузера вместо создания отдельного документа. Откройте консоль разработчика на любой странице браузера. Введите в консоль (можно скопировать и вставить) приведенное выше определение функции, убедитесь в отсутствии ошибок.

Этап 2: вызов и использование функции.

Попробуем вызвать функцию. Наберем в консоли

```
cube(2)
```

и нажмем «`Enter`». В качестве ответа в консоли появляется результат «`8`». Этот результат, например, можно сохранить в переменной «`y`»:

```
y = cube(2)
```

или использовать в расчете

```
2 * cube(2)
```

или передать как аргумент в другую функцию

```
alert( cube(2) )
```

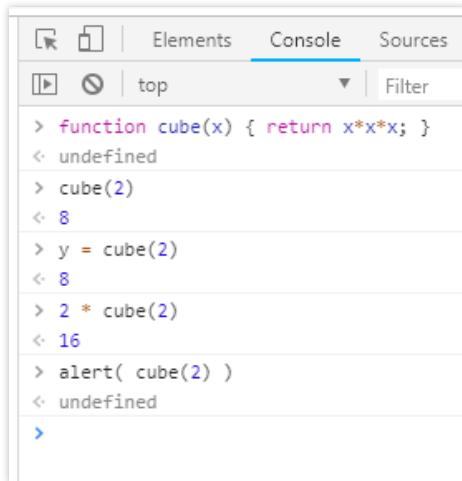


Рисунок 5

Ключевое слово «`return`», если сказать очень упрощенно, подставляет рассчитанный справа от него результат в то место программы, в котором был произведен вызов функции. Дальнейший код будет использовать полученный результат так, как будто вызова функции не было, а в данном месте кода было бы записано точно такое же значение.

В завершение первого знакомства с функциями приведем реализацию банковского округления. Сравните следующий код с примером из раздела «условия»:

```

function bankerRound(x) {
    if(Math.round(x)%2 == 0)
        return Math.round(x);
    else{
        if(x<Math.round(x) )
            return Math.round(x)-1;
        else

```

```
        return Math.round(x)+1;
    }
}
```

В отличие от исходного примера, вместо расчета итогового значения «**bx**», мы применили несколько команд «**return**». Такой прием может применяться для ускорения работы функции — когда нужное значение уже получено, нет необходимости его сохранять в отдельную переменную и анализировать дальнейшие условия. Можно завершить работу функции и передать в точку вызова итоговый результат.

Ведите или скопируйте определение функции в консоль браузера. Попробуйте ее вызывать с различными значениями аргумента

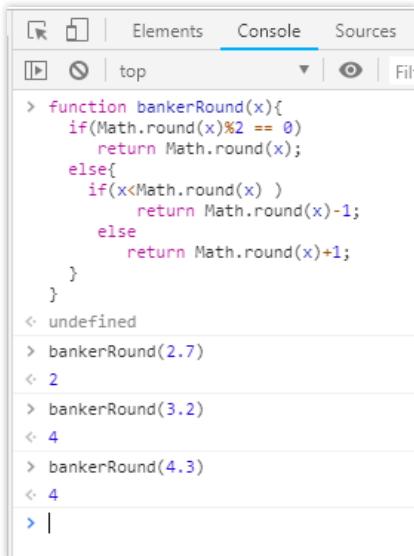


Рисунок 6

Найдите ошибку. Следующая функция должна вернуть строку «**odd**» если переданный в нее аргумент является нечетным числом или «**even**» — если четным. Правильным ли является следующее определение для такой функции?

```
function getParity(x) {  
    if(x % 2 == 0)  
        parity = "even";  
    else  
        parity = "odd";  
}
```

(В функции пропущена инструкция возврата значения. В конце тела должно быть указано «**return parity**». Либо в каждой ветке условного оператора вместо «**parity=**» можно написать «**return**».)

Задание для самостоятельной работы

1. Создайте функцию `sayError()`, которая будет выводить (при помощи диалогового окна `alert`) сообщение с текстом «`Some error occurred!`».
2. Создайте функцию `showError(x)`, которая будет выводить (при помощи диалогового окна `alert`) сообщение с текстом «`Error X occurred!`», где `X` — текст из аргумента функции (например, вызов `showError('Out of memory')` должен вывести сообщение «`Error Out of memory occurred!`»).
3. Создайте функцию `createHeaders(N)`, которая создаст на странице `N` заголовков второго уровня (`<h2>`) с надписями `Header1, Header2 ... HeaderN`.
4. Создайте функцию `checkPassword(x)`, которая вернет значение `true`, если в качестве аргумента в нее будет передан допустимый пароль (одно из значений «`Step`», «`Web`» или «`JavaScript`»). Иначе функция должна вернуть `false`.
5. Создайте функцию определения знака числа `sign(x)`, которая вернет значение `-1`, если аргумент «`x`» — отрицательное число, `1` — если положительное, `0` — если аргумент «`x`» равен нулю.
6. Предложите имя (согласно правилам именования) и создайте функцию, которая будет возвращать названия дней недели по их номеру: `0-Sunday, 1-Monday, 2-Tuesday, 3-Wednesday, 4-Thursday, 5-Friday, 6-Saturday`.



Unit 4. ФУНКЦИИ

© Денис Самойленко.

© Компьютерная Академия «Шаг», www.itstep.org.

Все права на охраняемые авторским правом фото-, аудио- и видеопротивления, фрагменты которых использованы в материале, принадлежат их законным владельцам. Фрагменты произведений используются в иллюстративных целях в объеме, оправданном поставленной задачей, в рамках учебного процесса и в учебных целях, в соответствии со ст. 1274 ч. 4 ГК РФ и ст. 21 и 23 Закона Украины «Про авторське право і суміжні права». Объем и способ цитируемых произведений соответствует принятым нормам, не наносит ущерба нормальному использованию объектов авторского права и не ущемляет законные интересы автора и правообладателей. Цитируемые фрагменты произведений на момент использования не могут быть заменены альтернативными, не охраняемыми авторским правом аналогами, и как таковые соответствуют критериям добросовестного использования и честного использования.

Все права защищены. Полное или частичное копирование материалов запрещено. Согласование использования произведений или их фрагментов производится с авторами и правообладателями. Согласованное использование материалов возможно только при указании источника.

Ответственность за несанкционированное копирование и коммерческое использование материалов определяется действующим законодательством Украины.