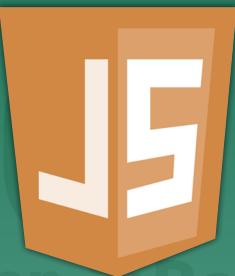


Разработка клиентских сценариев с использованием JavaScript и библиотеки jQuery



Unit 2

Взаимодействие с пользователем

Contents

Ввод/вывод данных. Диалоговые окна	3
Условия	10
Что такое условие?	10
If	14
If else.....	17
Тернарный оператор ?	21
Switch	23
Задание для самостоятельной работы	29

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

Ввод/вывод данных. Диалоговые окна

Одной из основных функций практически любой компьютерной программы является управление разнообразными потоками данных. Одни данные поступают в программу «извне», другие данные «выходят» из программы. Процессы передачи данных в программу и получения данных от нее называют «вводом/выводом данных» (англ. — *Input/Output, IO*).

Под вводом данных обычно подразумеваются процессы чтения данных из файла, получение их со сканнера, а также набор этих данных пользователем на клавиатуре, рисование их световым пером или диктовка на микрофон. Вывод данных — процесс обратного потока информации, обработанной программой: запись файлов, печать на различных устройствах, отображение на мониторах или проекторах, воспроизведение звука динамиками или колонками.

Частной задачей ввода/вывода является обеспечение взаимодействия программы с пользователем. В простейшем случае от пользователя ожидается ввод с клавиатуры некоторых данных, обработка их и отображение полученных результатов. В JavaScript существует несколько способов как получения данных от пользователя, так и вывода результатов их анализа. На данном этапе рассмотрим возможности, которые предоставляют для этих задач диалоговые окна.

Диалоговые окна — это дополнительные небольшие окна, которые появляются на веб-странице и сообща-

ют пользователю некоторую информацию, просят подтверждения действий или ввода некоторых данных.

Простейшее диалоговое окно предназначено для выдачи текстового сообщения пользователю. Такое окно создается командой «`alert()`» в скобках которой указывается значение для отображения. Это может быть простой текст, взятый в кавычки «`alert("Message")`», числовое значение «`alert(10)`» либо имя переменной «`alert(x)`». В последнем случае в окне будет выведено содержимое переменной.

Быстрее всего проверить функциональность диалогового окна можно из консоли браузера. Откройте консоль на любой вкладке браузера и введите команду «`alert("Message")`». Нажмите «Enter» и на странице появится дополнительное окно с сообщением:

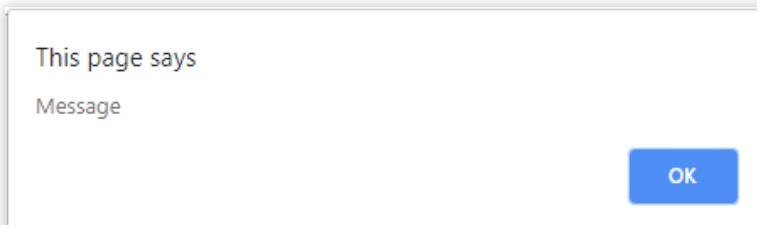


Рисунок 1

Обратите внимание, что вернуться в консоль не получается, пока окно-сообщение не будет закрыто. Такое поведение диалоговых окон называется «модальным» — блокирующим родительское окно на время работы диалога.

После нажатия кнопки «OK» сообщение закрывается и в консоли появляется результат выполнения введенной команды — «`undefined`». Это означает, что на самом деле никакого результата не возвращается. Команда «`alert`»

только оповещает пользователя и не получает от него никакого отклика.

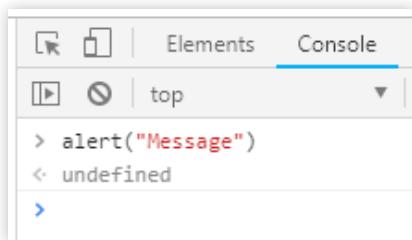


Рисунок 2

Если же отклик от пользователя требуется для коррекции дальнейшего выполнения программы, применяется команда «`confirm()`». Эта команда также выводит сообщение, но на диалоговом окне присутствуют две кнопки «OK» и «Cancel» (текст кнопок может отличаться в зависимости от языковых настроек браузера). Введите в консоли команду `confirm("Continue?")`. После нажатия кнопки «Enter» должно появиться диалоговое окно как на рисунке

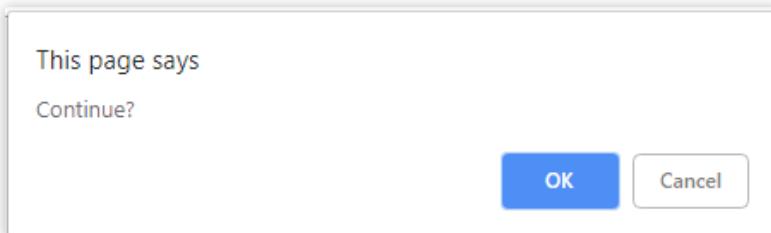


Рисунок 3

Убедитесь, что данное диалоговое окно также является модальным, то есть другие команды не выполняются, пока окно не закрыто. Нажмите «OK», окно закроется

и в консоли появится результат «`true`». Повторите ввод команды «`confirm("Continue?")`» (для быстрого повтора ранее введенных команд нажмите стрелку «вверх» на клавиатуре). На этот раз нажмите «`Cancel`». В таком случае результат выполнения будет «`false`».

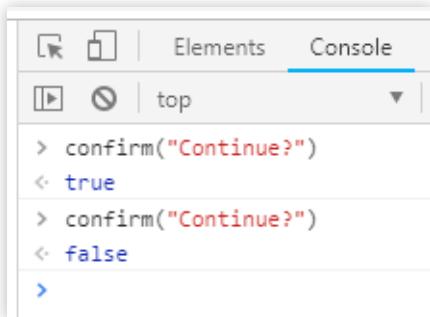


Рисунок 4

Анализируя полученные результаты несложно сделать вывод, что окно, созданное командой «`confirm`» возвращает значение типа «`Boolean`», при этом кнопке «`OK`» соответствует значение «`true`», а кнопке «`Cancel`» — «`false`». Обычно подобные окна используются для подтверждения пользователем некоторых действий или отказа от них.

Для того, чтобы получить от пользователя большее количество данных, предусмотрено диалоговое окно «`prompt`». Внешне оно похоже на окно «`confirm`», но содержит дополнительное поле для ввода данных. Команда «`prompt`» принимает два аргумента — сообщение пользователю и текст, который будет отображен в строке ввода изначально. Исследуем возможности этого диалогового окна — введем в консоли браузера «`prompt("Code for this`

`operation", "")», нажмем «Enter» и увидим новое окно с сообщением, переданным как аргумент команды и пустой строкой ввода (благодаря второму аргументу команды — пустым кавычкам).`

Как и все предыдущие диалоговые окна, окно «prompt» также является модальным, в чем несложно убедиться, попытавшись ввести в консоль новую команду. Введем в строке ввода некоторые данные.

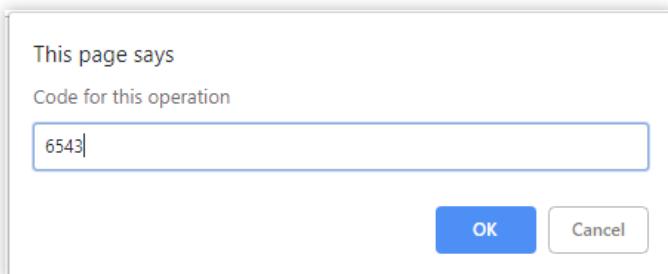


Рисунок 5

После ввода данных можно нажать кнопку «OK» или клавишу «Enter» на клавиатуре. Окно закроется и в консоли появится результат, представляющий введенную нами строку. Обратите внимание, что хотя в примере на рис. выше мы вводили число, результатом работы команды «prompt» все равно является строка, что видно по наличию кавычек в ответе консоли. Вспомните, как в этом можно убедиться при помощи оператора «typeof», и проведите эту проверку.

Повторите команду «`prompt("Code for this operation", "")`» еще раз (напоминаем, повтор команды — стрелка вверх). На этот раз нажмите «Cancel». В качестве возвращенного результата получим значение «`null`».

```

> prompt("Code for this operation","");
< "6543"
> prompt("Code for this operation","");
< null
>

```

Рисунок 6

Это значение может использоваться для программного контроля того, что пользователь отменил ввод данных. Однако, браузер «Safari» в случае отмены ввода возвращает не «`null`», а пустую строку. Об этом следует помнить при составлении универсальных (кроссбраузерных) сценариев.

С учетом того, что пустая строка не несет в себе никаких данных, нет принципиальной разницы, какая кнопка была нажата — «OK» или «Cancel», если данных все равно нет. С точки зрения логики взаимодействия с пользователем, пустая строка и значение «`null`» эквивалентны и означают одно и то же: пользователь не ввел данные для программы. Поэтому обрабатывать пустую строку и «`null`» вполне логично в одном блоке команд.

Задание: напишите логическое выражение для проверки того, что переменная «`x`» содержит значение «`null`» либо пустую строку.

Еще одной особенностью поведения относительно команды «`prompt`» обладает браузер «Internet Explorer». Если в нем не указать второй аргумент команды записав, например, «`prompt("Code for this operation")`», то в результате

в строке ввода изначально будет присутствовать надпись «`undefined`». В других браузерах строка ввода остается пустой при отсутствии второго аргумента, в том числе и в наследнике «IE» — браузере «Edge». Хотя «IE» теряет популярность все больше и больше, лучше не забывать указывать второй аргумент для команды «`prompt`».

Также следует отметить, что использование диалоговых окон в современных сайтах практически отсутствует в силу их скромного дизайна и невозможности его украсить, в т.ч. даже просто увеличить шрифт. Для поддержания общего оформления страницы можно создавать собственные средства диалога с пользователем, что обычно и делается на серьезных ресурсах. Стандартные диалоговые окна находят применение на этапе отладки сценариев, когда полный дизайн сайта еще не готов.

Условия

Разобравшись с тем, как можно организовать взаимодействие программы с пользователем при помощи диалоговых окон, перейдем к рассмотрению способов реакции программы на данные, полученные от пользователя.

При помощи команды `prompt("Code for this operation","")` программа может получить введенный пользователем код подтверждения некоторого действия и выполнить это действие, но только если полученный код правильный. Если же введенный код окажется неправильным, то действие будет считаться неподтвержденным и выполниться не должно. Можно вывести предупреждение об ошибочном коде, а можно просто отказаться от дальнейшей работы программы.

Таким образом, при составлении программ нам нужен инструмент, позволяющий выполнять какие-то инструкции только при соблюдении определенных критериев, а при отклонении от них либо выполнять другие инструкции, либо вообще ничего не выполнять. Такой инструмент реализуют условия и условные операторы.

Что такое условие?

Под условием (*англ. — condition*) в программировании подразумевается особый вид выражений, приводящих к одному из двух возможных результатов. В языках программирования, поддерживающих тип данных «`Boolean`» (в частности в `JavaScript`), эти результаты соответствуют константам `«false»` и `«true»`. Альтернативно, можно говорить «условие ложно» либо «условие истинно». В старых

языках, не имеющих поддержки типа «Boolean», результатами условий считались значения «0» или «1».

Много хороших примеров условных выражений предоставляет современный маркетинг: «при покупке на сумму свыше X вы получите скидку 10%», «при заказе 3-х товаров доставка бесплатно», «при заправке полностью бака — кофе в подарок». Каждое выражение имеет два исхода — либо вы получите указанное предложение, либо вам в этом предложении будет отказано. А какой из исходов будет реализован, определяет предшествующее условие, которое может быть либо выполненным (истинным), либо нет (ложным).

Также и в программировании, основная цель условия состоит в разделении программы на два пути — по одному из них программа пойдет, если условие истинно, по другому — в противном случае. Имеется в виду, что в программе будет два блока команд, и какой из них выполнится при работе программы, будет известно после проверки дополнительных условий. Как вариант, второй блок может быть пустым (то есть отсутствовать), в таком случае, при ложном условии ничего выполняться не будет.

В графическом представлении алгоритмов условие изображается в виде ромба, внутри которого записывается условное выражение. Из боковых углов ромба выходят две стрелки с подписями «+» и «-». Они определяют пути дальнейшего выполнения программы в зависимости от истинности (+) или ложности (-) данного условия. Алгоритмические схемы или аналогичные им UML диаграммы улучшают наглядность при разработке или анализе программ.

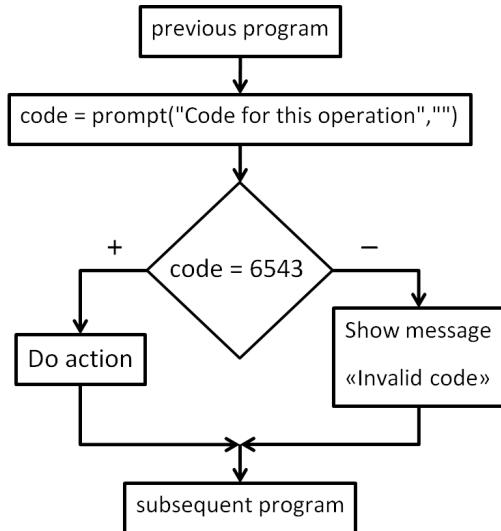


Рисунок 7

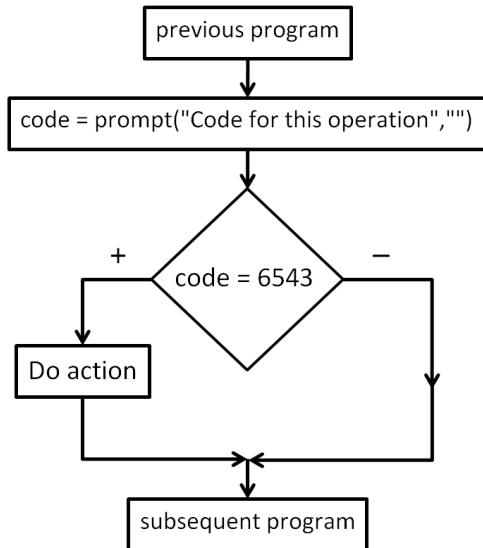


Рисунок 8

Обратим внимание на то, что оба блока команд должны быть описаны на этапе создания программы, и в момент ее запуска возможны оба пути выполнения. Ведь мы заранее не знаем, введет пользователь правильный код или нет.

В практическом программировании условия обычно применяются для:

- Подтверждения действий, авторизации (проверки логина и пароля), разблокировки программ.
- Проверки ограничений: на возраст пользователя, на время суток, на день недели.
- Проверки правильности пользовательского ввода. Очень часто, особенно в Веб-разработке, от пользователя требуется адрес электронной почты или номер телефона. При вводе этих данных пользователь может опечататься, забыть переключить раскладку клавиатуры или нарочно исказить вводимые данные. Каждое из вводимых значений должно быть проверено отдельным условием.
- Проверки успешной работы других частей программы, если от них зависят дальнейшие процессы, проверки на отсутствие ошибок расчетов или преобразований.

Обычно, условие состоит из сравнений и отношений, объединённых логическими операциями. В условиях могут присутствовать и другие операции или выражения, но итоговым результатом условия все равно должны быть «ложь» или «истина». Больше примеров условий будет приведено в последующих разделах урока.

Еще несколько слов о различной терминологии.

- Поскольку условное выражение имеет значение (истина или ложь), его можно назвать «условной операцией» (помним, операциями называют выражения, которые приводят к некоторому результату).
- В отдельных языках программирования (например, в языке PROLOG), а также в математической логике применяется термин «предикат». В различных источниках можно увидеть варианты «предикатное выражение» или «предикативное выражение».
- В память об одном из основателей математической логики — о Джордже Буле (*англ. George Boole*) условия также называют «булевыми выражениями». Иногда обобщенно — «логическими выражениями».

Все эти термины являются синонимами и могут применяться различными авторами учебников или статей для обозначения одних и тех же программных конструкций — условий.

If

Основным средством, позволяющим обработать условие, то есть выполнить некоторую часть программы в зависимости от результата условного выражения, является оператор «**if**». Его называют условным оператором или оператором ветвления, поскольку в результате проверки некоторого условия этот оператор обеспечит дальнейшее выполнение программы по тому или иному пути.

Синтаксис оператора выглядит следующим образом. После слова «**if**» в круглых скобках записывается условие

и, после скобок, — выражение (инструкция), которое нужно выполнить по результатам проверки условия.

```
if(condition)
    statement;
```

Выражение будет обработано и выполнено лишь в том случае, если условие приведет к результату «**true**», то есть если условие «выполняется» (истинно).

Если после проверки условия необходимо выполнить несколько инструкций, то их объединяют группирующим оператором «**{}**» (фигурные скобки)

```
if(condition) {
    statement1;
    statement2;
    statement3;
}
```

Для того чтобы проверить, как работает условный оператор, введите в консоль браузера выражение

```
if(1>0) alert("Yes")
```

В нем в качестве условия используется отношение «**1>0**», а инструкцией является рассмотренное нами ранее диалоговое окно-сообщение с текстом «**Yes**». Поскольку условие изначально истинно, команда «**alert("Yes")**» должна быть выполнена. Нажмите «**Enter**» и убедитесь в появлении сообщения. После закрытия этого сообщения в консоли появится надпись «**undefined**». Вспомните, что это означает?

Поменяйте условное выражение на другое, используя рассмотренные в предыдущих разделах примеры, и про-

верьте при каких случаях сообщение «Yes» появляется, а в каких нет.

Приведем несколько практических примеров, базирующихся на условном операторе.

Задание: в переменной «`x`» хранится разница вчерашнего и сегодняшнего курса валюты. Нас интересует только величина отклонения, то есть если значение отрицательное, то нужно убрать знак. Другими словами, нужно определить модуль числа «`x`».

Решение: при помощи условного оператора это сделать проще и быстрее, чем через библиотеку математических функций. Если число «`x`» меньше нуля, применим оператора смены знака «`-x`» и результат сохраним в той же переменной (`x = -x`):

```
if (x<0)
    x = -x;
```

Задание: необходимо проверить, была ли переменная «`x`» определена ранее в программе. Если нет, то нужно присвоить ей значение «`0`».

Решение: мы уже упоминали способ проверки на то, что переменная не была определена (см. раздел «Оператор `typeof`»), поместим его в условный оператор:

```
if(typeof x =='undefined')
    x = 0;
```

Задание: в переменной «`x`» хранится сумма покупки. Если она превышает `1000`, уменьшить ее на скидку в `10%`.

Решение: уменьшение на 10% эквивалентно вычитанию из числа одной десятой его части. То есть «**x**» уменьшенный на 10% это «**x – x/10**» или «**x – 0.9x**». После вычитания, результат нужно сохранить в той же переменной. С учетом этого запишем:

```
if(x>1000)
    x = x - 0.9*x
```

If else

В случае если по итогам проверки условия нужно выполнить два разных блока команд, применяется расширенная (или полная) форма условного оператора, содержащая две секции инструкций. Записывается полная форма условного оператора следующим образом:

```
if(condition)
    statement_if_true;
else
    statement_if_false;
```

Если результат проверки условия «**condition**» будет истинным, выполнится первое выражение «**statement_if_true**». В противном случае — второе «**statement_if_false**». Выделяя термин «полная» форма условного оператора отметим, что рассмотренная в предыдущем разделе форма называется сокращенной.

Полностью аналогично сокращенной форме, для выполнения нескольких выражений в полной форме условного оператора необходимо применить к ним групп-

пирующий оператор «{}». Для лучшей читаемости программы при использовании группировки рекомендуется добавлять одинаковые отступы слева перед сгруппированными выражениями.

```
if(condition) {
    statement_if_true1;
    statement_if_true2;
    statement_if_true3;
}

else {
    statement_if_false1;
    statement_if_false2;
    statement_if_false3;
}
```

Приведем несколько практических примеров использования полной формы условного оператора:

Задание: в переменной «`x`» хранится некоторое число. Необходимо проверить его четность и результат сохранить в переменной «`parity`»

Решение: четное число делится на `2` нацело. То есть имеет нулевой остаток от деления на `2`. В формализме JavaScript остаток от деления вычисляется оператором «`%`». Условие на четное значение будет выглядеть как «`x % 2 == 0`». В противном случае, число является нечетным

```
if(x % 2 == 0)
    parity = "even";
else
    parity = "odd";
```

Условные операторы могут быть вложенными один в другой. То есть в любом из условных блоков могут быть использованы новые условные операторы со своими блоками. В них также могут быть условные операторы — ограничений на степень вложенности нет. Это позволяет проверять условия одно за другим, обрабатывая на каждое из них отдельно.

В качестве примера рассмотрим следующую задачу. Банковское округление (англ. *banker's rounding*) обозначает округление к ближайшему чётному числу. То есть 2.7 округляется до 2, а 3.2 до 4 (хотя при обычном округлении оба значения привели бы к результату 3). В переменной «`x`» хранится дробное число, нужно написать программу, реализующую банковское округление этого числа и сохраняющую результат в переменной «`bx`».

Решение: во-первых, попробуем использовать обычное округление `Math.round(x)` и проверим, может мы сразу получим четное число, тогда его нужно просто сохранить в переменной «`bx`»:

```
if( Math.round(x) % 2 == 0 )
    bx = Math.round(x)
```

Если же результат обычного округления нечетен, то у нас есть два возможных варианта: а) число было округлено вверх, например 2,7 до тройки или б) число было округлено вниз, как 3,2 к той же тройке. В первом случае от результата округления надо вычесть единицу, получив меньшее из четных чисел (2,7 округляется к двойке). Во втором варианте единицу к результату нужно добавить

([3,2](#) округляется к четверке). Это можно реализовать вторым условным оператором, вложенным в блок «[else](#)» первого оператора. Итоговый код, решающий задачу, примет следующий вид:

```
if( Math.round(x) % 2 == 0 )
    bx = Math.round(x)
else {
    if( x < Math.round(x) )
        bx = Math.round(x) - 1
    else
        bx = Math.round(x) + 1
}
```

Благодаря условному оператору мы формируем множество путей работы программы, то есть задаем ее поведение, реакцию на изменяющееся окружение или действия пользователя. При сохранении постоянства кода программы, реальный ход его выполнения может значительно отличаться для разных ситуаций из-за того, что ветвления на каждом из условий приведут к различному набору действий. Начинаясь с одной и той же первой строки кода, дальнейший путь и место окончания заранее непредсказуемы и определяются в процессе работы программы.

Отдельно обратим внимание на то, что в JavaScript, за счет динамического преобразования типов, в качестве аргумента для оператора «[if](#)» может использоваться не только условное выражение, но и любая другая операция (например, арифметическая). В таком случае ее результат будет автоматически преобразовываться к типу «[Boolean](#)» и анализироваться итог такого преобразования. К «ложному» результату преобразуются значения «[0](#)»

(числовой ноль), «`null`», «`undefined`» или пустая строка. Другие результаты считаются истинными.

Например, в рассмотренной выше проверке на четность используется условие `(x % 2 == 0)`. Так как остаток от деления на 2 может быть только 1 или 0, можно воспользоваться динамическим преобразованием типов, записав условный оператор как `«if(x % 2)»` — без сравнения с нулем. Только следует отметить, что при значении «`0`» будет выполнен блок `«else»`, тогда как полное сравнение `(x % 2 == 0)` в этом же случае выполняет основной блок. То есть по сравнению с предыдущим примером в сокращенном варианте условные блоки нужно поменять местами.

```
if(x % 2)
    parity = "odd";
else
    parity = "even";
```

Тернарный оператор ?

Довольно часто в программировании возникают ситуации, когда по результату проверки какого-то условия в некоторую переменную нужно поместить то или иное значение. Снова вернемся к примеру проверки числа `«x»` на четность, в результате которой переменная `«parity»` принимала одно из значений: `«even»` или `«odd»`.

По сути, в переменную `«parity»` присваивается одно из двух значений в зависимости от условия проверки на четность переменной `«x»`. Читаемость программы улучшила бы конструкция, которая начиналась с записи `«parity=»`, после чего следовало бы выражение выбора значения.

Для реализации такой возможности был разработан тернарный оператор «?:». С его помощью приведенный блок команд можно записать как:

```
parity = x % 2 == 0 ? "even" : "odd"
```

Для лучшей разделимости составных частей тернарного оператора, условие берут в круглые скобки, подобно тому, как это делается в условном операторе. Согласитесь, следующее выражение выглядит более понятным, чем предыдущее:

```
parity = (x % 2 == 0) ? "even" : "odd"
```

Выражение начинается с «**parity**=». Это сразу подсказывает нам, что результатом операции будет присвоение переменной «**parity**» какого-то значения. Далее указывается условие «**x % 2 == 0**», что говорит о присваивании в зависимости от результата условия. Затем следуют два значения, разделенными символами «?» и «:», составляющими суть тернарного оператора. После символа «?» следует значение, которое будет возвращено при истинном результате условия; после символа «:» — при ложном результате.

Вместо конкретных значений («**even**» и «**odd**») возможно использовать выражения, приводящие к нужным результатам (операции). Правда, при этом читаемость тернарной инструкции может быть хуже, чем у аналогичного условного оператора, особенно при высокой сложности выражений.

Возможности тернарного оператора в JavaScript пре-восходят простое разделение значений при присваива-

нии. В состав оператора можно включать несколько операций, разделяя их запятыми. Однако это значительно ухудшает читаемость кода и крайне нежелательно к использованию на практике. Здесь мы эти возможности раскрывать не будем, следуя рекомендациям применять тернарный оператор, вместо условного, только для улучшения итоговой читаемости кода.

Switch

Ограниченный выбор одного из двух путей, реализуемый условным оператором, делает его неудобным для обработки множественных условий. С одной стороны, существует возможность вкладывать условные операторы один в другой, перебирая все варианты ветвления один за другим. С другой стороны, хотелось бы иметь возможность упростить подобный перебор.

Приведем простой пример. В программе мы получаем аббревиатуру протокола (HTTP, HTTPS, FTP) и хотим сформировать ее расшифровку, а также предусмотреть случай несовпадения полученного протокола ни с одним из этих значений. С применением вложенного ветвления это можно сделать следующим образом:

```
if(protocol == "HTTP") description =
    "Hypertext transfer protocol";
else if(protocol == "HTTPS") description =
    "Secure hypertext transfer protocol";
else if(protocol == "FTP") description =
    "File transfer protocol";
else description = "Unsupported protocol";
```

Очевидно, что с увеличением множества возможных значений протоколов, вложенность условий также возрастает, ухудшая читаемость кода и удобство его оформления в целом.

Для того чтобы упростить подобный множественный анализ применяется оператор «**switch**». При помощи него поставленная выше задача может быть решена как:

```
switch(protocol) {  
    case "HTTP":  
        description = "Hypertext transfer protocol";  
        break;  
    case "HTTPS":  
        description = "Secure hypertext  
                        transfer protocol";  
        break;  
    case "FTP":  
        description = "File transfer protocol";  
        break;  
    default :  
        description = "Unsupported protocol";  
}
```

Этот код читается гораздо лучше, да и выглядит понятнее, чем приведенный ранее вариант с вложенными условными операторами.

В круглых скобках после оператора «**switch**» указывается переменная или выражение, которое должно быть проверено на соответствие некоторому значению. Далее следуют фигурные скобки, в которых каждое из условий записывается после ключевого слова «**case**». Оператор сравнения «**==**» при этом опускается, приводится только само значение для сравнения.

Отдельным блоком выступает секция «`default`», которая будет выполнена, если ни одно из перечисленных в «`case`» выражений не будет обработано. Его можно сравнить с неким итоговым «`else`» по аналогии с условным оператором.

Исторически сложилось так, что оператор «`switch`» выполнял все инструкции, следующие далее за подходящим «`case`». Если значение в определенном «`case`» будет равно переменной из оператора «`switch`», то все нижеследующие команды данного блока, а затем все остальные команды всех последующих блоков «`case`», в том числе и команды блока «`default`», будут выполнены один за другим в порядке описания. В JavaScript сохранен это формализм и следует об этом помнить при его использовании.

С одной стороны, это позволяет не группировать команды при помощи фигурных скобок, сокращая разметку. С другой стороны, требует применение инструкции прерывания «`break`», если дальнейшее выполнение кода не должно быть реализовано. Повторимся — это дань традициям, и практически во всех языках программирования оператор «`switch`» работает подобным образом.

Приведем пример, иллюстрирующий описанное поведение оператора «`switch`». Пусть задача состоит в том, чтобы сформировать строку, начиная с введенного пользователем значения переменной «`x`» и заканчивая значением «`5`». Добавим ограничение, что значение «`x`» не должно быть больше 5 и меньше 1.

Данную задачу можно решить следующим способом. Для ее решения создайте новый HTML документ, наберите или скопируйте в него следующее содержимое (код также доступен в папке *Sources* — файл `js1_5.html`)

```
<!doctype html>
<html>
    <head>
    </head>
    <body>
        <script>
            x = +prompt("Input x=");
            var str = "";
            switch(x) {
                case 1: str += "1";
                case 2: str += "2";
                case 3: str += "3";
                case 4: str += "4";
                default: str += "5";
            }
            alert(str);
        </script>
    </body>
</html>
```

Поясним, как работает данная программа. Вначале у пользователя запрашивается первое значение для задачи при помощи диалогового окна «`prompt`». Как нам известно, значение из окна «`prompt`» будет передано в строковом типе (см. предыдущий раздел), поэтому мы применим его преобразование к числовому типу при помощи унарного оператора «`+`». Полученный результат сохраняем в переменной «`x`» (`x = +prompt("Input x")`).

Далее создается переменная «`str`» и ей присваивается пустая строка. В операторе «`switch`» проводится анализ переменной «`x`». Если значение переменной равно `1`, то срабатывает самый первый «`case`» и в строку «`str`» дописывается символ «`1`» при помощи операции добавления «`+=`».

Согласно принципу работы оператора «`switch`», далее будут выполнены все блоки команд после первого успешного блока «`case`», то есть к строке «`str`» будут последовательно дописаны символы от «`2`» до «`5`» за счет выполнения всех дальнейших команд из других блоков.

Если значение «`x`» равно `2`, то совпадение будет найдено во втором «`case`» и, по аналогии, в строку будут последовательно собраны все остальные символы за счет команд последующих блоков. Команда из первого блока «`case`» при этом выполнена не будет. Выполняются только те команды, которые описаны ниже подходящего блока «`case`».

В конце программы сформированная строка выводится при помощи окна-сообщения «`alert`».

Сохраните файл, откройте его при помощи браузера. В появившемся диалоговом окне впишите число от `1` до `5` и нажмите «`OK`». Убедитесь в появлении нового сообщения с числовой последовательностью.

Обратите внимание, что введенные пользователем данные мы преобразовывали к числу. Это необходимо потому, что в операторе «`switch`» используется только строгое сравнение «`==`» (см. раздел 13). Для того чтобы в этом убедиться, удалите оператор «`+`» перед «`prompt`», сохраните файл и обновите страницу в браузере. При любом вводе в первом диалоговом окне результатом работы будет «`5`» за счет выполнения блока «`default`», т.к. сравнение числовых данных, указанных в «`case`», и строкового результата диалога «`prompt`» всегда будет приводить к ложному результату.

Задача для самостоятельного решения. Пользователь вводит число «`x`». Считаем, что пользовательский

ввод ограничен значениями от **1** до **10**. Программа выводит числа от «**x**» до **5**, если **x≤5**, иначе — числа от «**x**» до **10**.

Найдите ошибку: программа должна запросить у пользователя число от 1 до 6 и вывести сообщение о четности числа (**odd** — нечетное, **even** — четное), либо "Out of range", если число выходит за указанный диапазон

```
<script>
    x = +prompt("Input number from 1 to 6");
    switch(x) {
        case 2:
        case 4:
        case 6: alert("even");
            break;
        case 1:
        case 3:
        case 5: alert("odd");
        default: alert("Out of range");
    }
</script>
```

(Перед вариантом «**default**» пропущен оператор **«break»**. Если будет введено нечетное число, то выведутся оба окна — **alert("odd")** и **alert("Out of range")**, т.к. в операторе **«switch»** выполняются все инструкции, указанные ниже подходящего условия).

Задание для самостоятельной работы

1. Напишите скрипт, который запрашивает у пользователя подтверждение некоторого действия (используя диалог `confirm`) и после его ответа выводит сообщение «Подтверждено» или «Отменено».
2. Напишите скрипт, который запрашивает у пользователя пароль подтверждения некоторого действия. Допускается три возможных пароля («`Step`», «`Web`» и «`JavaScript`»). После ввода пароля скрипт должен вывести сообщение «Подтверждено» или «Отменено».
3. Напишите скрипт, который запрашивает у пользователя число «`x`», проверяет его на принадлежность диапазону `0..100` и выводит соответствующее сообщение (например, `10` — принадлежит, `-10` — не принадлежит, `0` — принадлежит, `200` — не принадлежит).
4. Напишите скрипт, который запрашивает у пользователя два числа «`x`» и «`y`», сравнивает их величины и выводит одно из сообщений: «`x > y`», «`x < y`» или «`x=y`» в зависимости от введенных данных.
5. Напишите скрипт, который запрашивает у пользователя число «`x`», «ранжирует» его по диапазонам `0..100`, `101..200`, `201..300` и выводит сообщение о принадлежности или несоответствии ни одному из них (например, `30` принадлежит диапазону `0..100`; `250` — диапазону `201..300`; `-10` или `500` — ни одному).
6. Напишите скрипт, который запрашивает у пользователя цифру и выводит ее название: `0` — «ноль», `1` — «единица», `2` — «двойка» и т.д. Если переменная не является цифрой, выводится сообщение «не цифра».



Unit 2.

Взаимодействие с пользователем

© Денис Самойленко.

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

Все права на охраняемые авторским правом фото-, аудио- и видеопропизведения, 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.