# Конструкция switch

Конструкция switch заменяет собой сразу несколько if.

Это — более наглядный способ сравнить выражение сразу с несколькими вариантами.

## Синтаксис

Выглядит она так:

switch(x)

{

case 'value1': // if (x === 'value1')

...

[break]

case 'value2': // if (x === 'value2')

...

[break]

default:

...

[break]

}

• Переменная x проверяется на строгое равенство первому значению value1, затем второмуvalue2 и так далее.

• Если соответствие установлено — switch начинает выполняться от соответствующей директивы case и далее, до ближайшего break (или до конца switch).

При этом case называют вариантами switch.

• Если ни один case не совпал — выполняетcя (если есть) вариант default.

var a = 2+2;

switch (a)

{

case 3:

alert('Маловато');

break;

case 4:

alert('В точку!');

break;

case 5:

alert('Перебор');

break;

default:

alert('Я таких значений не знаю');

}

Будет выведено только одно значение, соответствующее 4. После чего break прервёт выполнение.

Если его не прервать — оно пойдёт далее, при этом остальные проверки игнорируются.

Например:

var a = 2+2;

switch (a)

{

case 3:

alert('Маловато');

case 4:

alert('В точку!');

case 5:

alert('Перебор');

default:

alert('Я таких значений не знаю');

}

В примере выше последовательно выполнятся три alert.

alert('В точку!');

alert('Перебор');

alert('Я таких значений не знаю');

В case могут быть любые выражения, в том числе включающие в себя переменные и функции.

Например:

var a = 1;

var b = 0;

switch(a)

{

case b+1:

alert(1);

break;

default:

alert('нет-нет, выполнится вариант выше')

}

## Группировка case

Несколько значений case можно группировать.

В примере ниже case 3 и case 5 выполняют один и тот же код:

var a = 2+2;

switch (a)

{

case 4:

alert('Верно!');

break;

case 3: // (\*)

case 5: // (\*\*)

alert('Неверно!');

break;

default:

alert('Я таких значений не знаю');

}

При case 3 выполнение идёт со строки (3) и идёт вниз до ближайшего break, таким образом проходя и то, что предназначено для case 5.

## Тип имеет значение

Следующий пример принимает значение от посетителя.

var arg = prompt("Введите arg?")

switch(arg)

{

case '0':

case '1':

alert('Один или ноль');

case '2':

alert('Два');

break;

case 3:

alert('Никогда не выполнится');

case null:

alert('Отмена');

break;

default:

alert('Неизвестное значение: ' + arg)

}

Что оно выведет при вводе чисел 0, 1, 2, 3? Подумайте и потом читайте дальше…

• При вводе 0 или 1 выполнится первый alert, далее выполнение продолжится вниз до первогоbreak и выведет второй alert('Два').

• При вводе 2, switch перейдет к case '2' и выведет Два.

• При вводе 3, switch перейдет на default. Это потому, что prompt возвращает строку '3', а не число. Типы разные. Switch использует строгое равенство ===, так что совпадения не будет.

• При отмене сработает case null.

# Функции

Зачастую, нам надо повторять одно и то же действие во многих частях программы.

Например, красиво вывести сообщение необходимо при приветствии посетителя, при выходе посетителя с сайта, еще где-нибудь.

Чтобы не повторять один и тот же код во многих местах, придуманы функции. Функции являются основными «строительными блоками» программы.

Примеры встроенных функций вы уже видели — это alert(message), prompt(message, default) иconfirm(question). Но можно создавать и свои.

## Объявление

Пример объявления функции:

function showMessage()

{

alert('Привет всем присутствующим!');

}

Вначале идет ключевое слово function, после него имя функции, затем список параметров в скобках (в примере выше он пустой) и тело функции — код, который вызывается при её вызове.

Объявленная функция доступна по имени, например:

function showMessage()

{

alert('Привет всем присутствующим!');

}

showMessage();

showMessage();

Этот код выведет сообщение два раза. Уже здесь видна главная цель создания функций: избавление от дублирования кода.

Если понадобится поменять сообщение или способ его вывода — достаточно изменить его в одном месте: в функции, которая его выводит.

## Локальные переменные

Функция может содержать локальные переменные, объявленные через var. Такие переменные видны только внутри функции:

function showMessage()

{

var message = 'Привет, я - Вася!'; // локальная переменная

alert(message);

}

showMessage(); // 'Привет, я - Вася!'

alert(message); // <-- будет ошибка, т.к. переменная видна только внутри

Блоки if/else, switch, for, while, do..while не влияют на область видимости переменных.

При объявлении переменной в таких блоках, она всё равно будет видна во всей функции.

Например:

function count()

{

for (var i=0; i<3; i++)

{

var j = i \* 2;

}

alert(i); // i=3, на этом значении цикл остановился

alert(j); // j=4, последнее значение, на котором цикл сработал, было i=2

}

Неважно, где именно в функции и сколько раз объявляется переменная. Любое объявление срабатывает один раз и распространяется на всю функцию.

Объявления переменных в примере выше можно передвинуть вверх, это ни на что не повлияет:

function count()

{

var i, j; // передвинули объявления var в начало

for (i=0; i<3; i++)

{

j = i \* 2;

}

alert(i); // i=3

alert(j); // j=4

}

## Внешние переменные

Функция может обратиться ко внешней переменной, например:

var userName = 'Вася';

function showMessage()

{

var message = 'Привет, я ' + userName;

alert(message);

}

showMessage(); // Привет, я Вася

Доступ возможен не только на чтение, но и на запись. При этом, так как переменная внешняя, то изменения будут видны и снаружи функции:

var userName = 'Вася';

function showMessage()

{

userName = 'Петя'; // (1) присвоение во внешнюю переменную

var message = 'Привет, я ' + userName;

alert(message);

}

showMessage();

alert(userName); // Петя, значение внешней переменной изменено функцией

Конечно, если бы внутри функции, в строке (1), была бы объявлена своя локальная переменная var userName, то все обращения использовали бы её, и внешняя переменная осталась бы неизменной.

Переменные, объявленные на уровне всего скрипта, называют «глобальными переменными».

Делайте глобальными только те переменные, которые действительно имеют общее значение для вашего проекта.

Пусть каждая функция работает «в своей песочнице».

Внимание: неявное объявление глобальных переменных!

В старом стандарте JavaScript существовала возможность неявного объявления переменных присвоением значения.

Например:

function showMessage()

{

message = 'Привет'; // без var!

}

showMessage();

alert(message); // Привет

В коде выше переменная message нигде не объявлена, а сразу присваивается. Скорее всего, программист просто забыл поставить var.

В современном стандарте JavaScript такое присвоение запрещено, а в старом, который работает в браузерах по умолчанию, переменная будет создана автоматически, причём в примере выше она создаётся не в функции, а на уровне всего скрипта.

Избегайте этого.

Здесь опасность даже не в автоматическом создании переменной, а в том, что глобальные переменные должны использоваться тогда, когда действительно нужны «общескриптовые» параметры.

Забыли var в одном месте, потом в другом — в результате две функции неожиданно друг для друга поменяли одну и ту же глобальную переменную.

## Параметры

При вызове функции ей можно передать данные, которые та использует по своему усмотрению.

Например, этот код выводит два сообщения:

function showMessage(from, text)

{ // параметры from, text

from = "\*\* " + from + " \*\*"; // здесь может быть сложный код оформления

alert(from + '\n\n' + text);

}

showMessage('Маша', 'Привет!');

showMessage('Маша', 'Как дела?');

Параметры копируются в локальные переменные функции.

В примере ниже изменение from в строке (1) не отразится на значении внешней переменной from(2), т.к. изменена была копия значения:

function showMessage(from, text)

{

from = '\*\*' + from + '\*\*'; // (1), красиво оформили from

alert(from + '\n\n' + text);

}

var from = 'Маша', msg = 'Привет!'; // (2)

showMessage(from, msg); // значения будут скопированы в параметры

alert(from); // перезапись в строке (1) не повлияет на внешнюю переменную

## Аргументы по умолчанию

Функцию можно вызвать с любым количеством аргументов.

Например, функцию показа сообщения showMessage(from, text) можно вызвать с одним аргументом:

showMessage("Маша");

Если параметр не передан при вызове — он считается равным undefined.

Такую ситуацию можно отловить и назначить значение «по умолчанию»:

function showMessage(from, text)

{

if (text === undefined)

{

text = 'текст не передан';

}

alert(from + ": " + text);

}

showMessage("Маша", "Привет!"); // Маша: Привет!

showMessage("Маша"); // Маша: текст не передан

При объявлении функции необязательные аргументы, как правило, располагают в конце списка.

Для указания значения «по умолчанию», то есть, такого, которое используется, если аргумент не указан, используется два способа:

1. Можно проверить, равен ли аргумент undefined, и если да — то записать в него значение по умолчанию. Этот способ продемонстрирован в примере выше.

2. Использовать оператор ||:

function showMessage(from, text)

{

text = text || 'текст не передан';

...

}

Второй способ считает, что аргумент отсутствует, если передана пустая строка, 0, или вообще любое значение, которое в булевом виде является false.

## Возврат значения

Функция может возвратить результат, который будет передан в вызвавший её код.

Например, создадим функцию calcD, которая будет возвращать дискриминант квадратного уравнения по формуле b2 - 4ac:

function calcD(a, b, c)

{

return b\*b - 4\*a\*c;

}

var test = calcD(-4, 2, 1);

alert(test); // 20

Для возврата значения используется директива return.

Она может находиться в любом месте функции. Как только до нее доходит управление — функция завершается и значение передается обратно.

Вызовов return может быть и несколько, например:

function checkAge(age)

{

if (age > 18)

{

return true;

} else

{

return confirm('Родители разрешили?');

}

}

var age = prompt('Ваш возраст?');

if (checkAge(age))

{

alert('Доступ разрешен');

} else

{

alert('В доступе отказано');

}

Директива return может также использоваться без значения, чтобы прекратить выполнение и выйти из функции.

Например:

function showMovie(age)

{

if (!checkAge(age))

{

return;

}

alert("Фильм не для всех"); // (\*)

// ...

}

В коде выше, если сработал if, то строка (\*) и весь код под ней никогда не выполнится, так как return завершает выполнение функции.

Значение функции без return и с пустым return

В случае, когда функция не вернула значение или return был без аргументов, считается что она вернула undefined:

function doNothing() { /\* пусто \*/ }

alert( doNothing() ); // undefined

Обратите внимание, никакой ошибки нет. Просто возвращается undefined.

Ещё пример, на этот раз с return без аргумента:

function doNothing()

{

return;

}

alert( doNothing() === undefined ); // true

## Выбор имени

Имя функции следует тем же правилам, что и имя переменной. Основное отличие — оно должно быть глаголом, т.к. функция — это действие.

Как правило, используются глагольные префиксы, обозначающие общий характер действия, после которых следует уточнение.

Функции, которые начинаются с "show" — что-то показывают:

showMessage(..) // префикс show, "показать" сообщение

Функции, начинающиеся с "get" — получают, и т.п.:

getAge(..) // get, "получает" возраст

calcD(..) // calc, "вычисляет" дискриминант

createForm(..) // create, "создает" форму

checkPermission(..) // check, "проверяет" разрешение, возвращает true/false

Это очень удобно, поскольку взглянув на функцию — мы уже примерно представляем, что она делает, даже если функцию написал совсем другой человек, а в отдельных случаях — и какого вида значение она возвращает.

Одна функция — одно действие

Функция должна делать только то, что явно подразумевается её названием. И это должно быть одно действие.

Если оно сложное и подразумевает поддействия — может быть имеет смысл выделить их в отдельные функции? Зачастую это имеет смысл, чтобы лучше структурировать код.

Но самое главное — в функции не должно быть ничего, кроме самого действия и поддействий, неразрывно связанных с ним.

Например, функция проверки данных (скажем, "validate") не должна показывать сообщение об ошибке. Её действие — проверить.

# Методы и свойства

Все значения в JavaScript, за исключением null и undefined, содержат набор вспомогательных функций и значений, доступных «через точку».

Такие функции называют «методами», а значения — «свойствами». Посмотрим на примеры.

## Пример: str.length, str.toUpperCase()

У строки есть свойство length, содержащее длину:

alert( "Привет, мир!".length ); // 12

Еще у строк есть метод toUpperCase(), который возвращает строку в верхнем регистре:

var hello = "Привет, мир!";

alert( hello.toUpperCase() ); // "ПРИВЕТ, МИР!"

Если через точку вызывается функция (toUpperCase()) — это называют «вызов метода», если просто читаем значение (length) — «получение свойства».

## Пример: num.toFixed

У чисел есть метод num.toFixed(n). Он округляет число num до n знаков после запятой, при необходимости добивает нулями до данной длины и возвращает в виде строки (удобно для форматированного вывода):

var n = 12.345;

alert( n.toFixed(2) ); // "12.34"

alert( n.toFixed(0) ); // "12"

alert( n.toFixed(5) ); // "12.34500"

Обращение к методам чисел

К методу числа можно обратиться и напрямую:

alert( 12.34.toFixed(1) ); // 12.3

Но если число целое, то будет проблема:

alert( 12.toFixed(1) ); // ошибка!

Ошибка произойдёт потому, что JavaScript ожидает десятичную дробь после точки.

Это — особенность синтаксиса JavaScript. Вот так — будет работать:

alert( 12..toFixed(1) ); // 12.0

## Вызов метода — через круглые скобки!

Обратите внимание, вызов метода идёт через круглые скобки. Вообще-то, можно попробовать и без них, но результат будет совсем другой.

Метод — это функция, привязанная к значению. Если обратиться к ней без скобок, то результатом будет сама функция.

Непонятно? Посмотрите, например, результат обращения к toUpperCase без скобок:

var hello = "Привет";

alert( hello.toUpperCase ); // function...

Этот код выводит значение свойства toUpperCase, которое является встроенной в язык функцией. Как правило браузер выводит его как-то так:"function toUpperCase() { [native code] }".

Для получения результата эту функцию необходимо вызвать, поэтому и обращение делается со скобками:

var hello = "Привет";

alert( hello.toUpperCase() ); // "ПРИВЕТ" (результат вызова)

# Строки

В JavaScript любые текстовые данные являются строками. Не существует отдельного типа «символ», который есть в ряде других языков.

Внутренним форматом строк, вне зависимости от кодировки страницы, является Юникод (Unicode).

## Создание строк

Строки создаются при помощи двойных или одинарных кавычек:

var text = "моя строка";

var anotherText = 'еще строка';

var str = "012345";

В JavaScript нет разницы между двойными и одинарными кавычками.

## Специальные символы

Строки могут содержать специальные символы. Самый часто используемый из таких символов — это перевод строки.

Он обозначается как \n, например:

alert('Привет\nМир'); // выведет "Мир" на новой строке

Есть и более редкие символы, вот их список:

Специальные символы

Символ Описание

\b Backspace

\f Form feed

\n New line

\r Carriage return

\t Tab

\uNNNN Символ в кодировке Юникод с шестнадцатиричным кодом NNNN. Например, \u00A9 — юникодное представление символа копирайт ©

Экранирование специальных символов

Если строка в одинарных кавычках, то внутренние одинарные кавычки внутри должны быть экранированы, то есть снабжены обратным слешем \', вот так:

var str = 'I\'m a JavaScript programmer';

В двойных кавычках — экранируются внутренние двойные:

var str = "I'm a JavaScript \"programmer\" ";

alert(str);

Экранирование служит исключительно для правильного восприятия строки JavaScript. В памяти строка будет содержать сам символ без '\'. Вы можете увидеть это, запустив пример выше.

Сам символ обратного слэша '\' является служебным, поэтому всегда экранируется, т.е пишется как\\:

var str = ' символ \\ ';

alert(str); // символ \

Заэкранировать можно любой символ. Если он не специальный, то ничего не произойдёт:

alert( "\a" ); // a

// идентично alert( "a" );

## Методы и свойства

Здесь мы рассмотрим методы и свойства строк, с некоторыми из которых мы знакомились ранее, в главе Методы и свойства.

## Длина length

Одно из самых частых действий со строкой — это получение ее длины:

var str = "My\n"; // 3 символа. Третий - перевод строки

alert(str.length); // 3

## Доступ к символам

Чтобы получить символ, используйте вызов charAt(позиция). Первый символ имеет позицию 0:

var str = "jQuery";

alert( str.charAt(0) ); // "j"

В JavaScript нет отдельного типа «символ», так что charAt возвращает строку, состоящую из выбранного символа.

В современных браузерах (не IE7-) для доступа к символу можно также использовать квадратные скобки:

var str = "Я - современный браузер!";

alert(str[0]); // "Я", IE8+

Разница между этим способом и charAt заключается в том, что если символа нет —charAt выдает пустую строку, а скобки — undefined:

alert( "".charAt(0) ); // пустая строка

alert( ""[0] ); // undefined, IE8+

## Вызов метода — всегда со скобками

Обратите внимание, str.length — это свойство строки, а str.charAt(pos) —метод, т.е. функция.

Обращение к методу всегда идет со скобками, а к свойству — без скобок.

## Изменения строк

Строки в JavaScript нельзя изменять. Можно прочитать символ, но нельзя заменить его. Как только строка создана — она такая навсегда.

Чтобы это обойти, создаётся новая строка и присваивается в переменную вместо старой:

var str = "строка";

str = str.charAt(3) + str.charAt(4) + str.charAt(5);

alert(str); // ока

## Смена регистра

Методы toLowerCase() и toUpperCase() меняют регистр строки на нижний/верхний:

alert( "Интерфейс".toUpperCase() ); // ИНТЕРФЕЙС

Пример ниже получает первый символ и приводит его к нижнему регистру:

alert( "Интерфейс".charAt(0).toLowerCase() ); // 'и'

## Поиск подстроки

Для поиска подстроки есть метод indexOf(подстрока[, начальная\_позиция]).

Он возвращает позицию, на которой находится подстрока или -1, если ничего не найдено. Например:

var str = "Widget with id";

alert( str.indexOf("Widget") ); // 0, т.к. "Widget" найден прямо в начале str

alert( str.indexOf("id") ); // 1, т.к. "id" найден, начиная с позиции 1

alert( str.indexOf("Lalala") ); // -1, подстрока не найдена

Необязательный второй аргумент позволяет искать, начиная с указанной позиции. Например, первый раз "id" появляется на позиции 1. Чтобы найти его следующее появление — запустим поиск с позиции 2:

var str = "Widget with id";

alert( str.indexOf("id", 2) ) // 12, поиск начат с позиции 2

Также существует аналогичный метод lastIndexOf, который ищет не с начала, а с конца строки.

## Взятие подстроки: substr, substring, slice.

В JavaScript существуют целых 3 (!) метода для взятия подстроки, с небольшими отличиями между ними.

**substring(start [, end])**

Метод substring(start, end) возвращает подстроку с позиции start до, но не включая end.

var str = "stringify";

alert(str.substring(0,1)); // "s", символы с позиции 0 по 1 не включая 1.

Если аргумент end отсутствует, то идет до конца строки:

var str = "stringify";

alert(str.substring(2)); // ringify, символы с позиции 2 до конца

**substr(start [, length])**

Первый аргумент имеет такой же смысл, как и в substring, а второй содержит не конечную позицию, а количество символов.

var str = "stringify";

str = str.substr(2,4); // ring, со 2й позиции 4 символа

alert(str)

Если второго аргумента нет — подразумевается «до конца строки».

**slice(start [, end])**

Возвращает часть строки от позиции start до, но не включая, позиции end. Смысл параметров — такой же как в substring.

## Отрицательные аргументы

Различие между substring и slice — в том, как они работают с отрицательными и выходящими за границу строки аргументами:

**substring(start, end)**

Отрицательные аргументы интерпретируются как равные нулю. Слишком большие значения усекаются до длины строки:

alert( "testme".substring(-2) ); // "testme", -2 становится 0

Кроме того, если start > end, то аргументы меняются местами, т.е. возвращается участок строки между start и end:

alert( "testme".substring(4, -1) ); // "test"

// -1 становится 0 -> получили substring(4, 0)

// 4 > 0, так что аргументы меняются местами -> substring(0, 4) = "test"

**slice**

Отрицательные значения отсчитываются от конца строки:

alert( "testme".slice(-2) ); // "me", от 2 позиции с конца

alert( "testme".slice(1, -1) ); // "estm", от 1 позиции до первой с конца.

Это гораздо более удобно, чем странная логика substring.

Отрицательное значение первого параметра поддерживается в substr во всех браузерах, кроме IE8-.

**Вывод.**

Самый удобный метод — это slice(start, end).

В качестве альтернативы можно использовать substr(start, length), помня о том, что IE8- не поддерживает отрицательный start.

<http://habrahabr.ru/post/51717/>