**Псевдомассив аргументов "arguments"**

В JavaScript любая функция может быть вызвана с произвольным количеством аргументов.

Например:

function go(a,b) {

alert("a="+a+", b="+b);

}

go(1); // a=1, b=undefined

go(1,2); // a=1, b=2

go(1,2,3); // a=1, b=2, третий аргумент не вызовет ошибку

**В JavaScript нет «перегрузки» функций**

В некоторых языках программист может создать две функции с одинаковым именем, но разным набором аргументов, а при вызове интерпретатор сам выберет нужную:

function log(a) {

...

}

function log(a, b, c) {

...

}

log(a); // вызовется первая функция

log(a, b, c); // вызовется вторая функция

Это называется «полиморфизмом функций» или «перегрузкой функций». В JavaScript ничего подобного нет.

**Может быть только одна функция с именем log, которая вызывается с любыми аргументами.**

А уже внутри она может посмотреть, с чем вызвана и по-разному отработать.

В примере выше второе объявление log просто переопределит первое.

## [Доступ к «лишним» аргументам](http://learn.javascript.ru/arguments-pseudoarray" \l "dostup-k-lishnim-argumentam)

Как получить значения аргументов, которых нет в списке параметров?

Доступ к ним осуществляется через «псевдо-массив» [arguments](https://developer.mozilla.org/en/JavaScript/Reference/functions_and_function_scope/arguments).

Он содержит список аргументов по номерам: arguments[0], arguments[1]…, а также свойство length.

Например, выведем список всех аргументов:

function sayHi() {

for (var i = 0; i < arguments.length; i++) {

alert( "Привет, " + arguments[i] );

}

}

sayHi("Винни", "Пятачок"); // 'Привет, Винни', 'Привет, Пятачок'

Все параметры находятся в arguments, даже если они есть в списке. Код выше сработал бы также, будь функция объявлена sayHi(a,b,c).

**Связь между arguments и параметрами**

**В старом стандарте JavaScript псевдо-массив arguments и переменные-параметры ссылаются на одни и те же значения.**

В результате изменения arguments влияют на параметры и наоборот.

Например:

function f(x) {

arguments[0] = 5; // меняет переменную x

alert( x ); // 5

}

f(1);

Наоборот:

function f(x) {

x = 5;

alert( arguments[0] ); // 5, обновленный x

}

f(1);

В современной редакции стандарта это поведение изменено. Аргументы отделены от локальных переменных:

function f(x) {

"use strict"; // для браузеров с поддержкой строгого режима

arguments[0] = 5;

alert( x ); // не 5, а 1! Переменная "отвязана" от arguments

}

f(1);

**Если вы не используете строгий режим, то чтобы переменные не менялись «неожиданно», рекомендуется никогда не изменять arguments.**

### [arguments – это не массив](http://learn.javascript.ru/arguments-pseudoarray" \l "arguments-eto-ne-massiv)

Частая ошибка новичков – попытка применить методы Array к arguments. Это невозможно:

function sayHi() {

var a = arguments.shift(); // ошибка! нет такого метода!

}

sayHi(1);

Дело в том, что arguments – это не массив Array.

В действительности, это обычный объект, просто ключи числовые и есть length. На этом сходство заканчивается. Никаких особых методов у него нет, и методы массивов он тоже не поддерживает.

Впрочем, никто не мешает сделать обычный массив из arguments, например так:

var args = [];

for (var i = 0; i < arguments.length; i++) {

args[i] = arguments[i];

}

Такие объекты иногда называют «коллекциями» или «псевдомассивами».

## [Пример: копирование свойств copy(dst, src1, src2...)](http://learn.javascript.ru/arguments-pseudoarray" \l "copy)

Иногда встаёт задача – скопировать в существующий объект свойства из одного или нескольких других.

Напишем для этого функцию copy. Она будет работать с любым числом аргументов, благодаря использованию arguments.

Синтаксис:

**copy(dst, src1, src2…)**

Копирует свойства из объектов src1, src2,... в объект dst. Возвращает получившийся объект.

Использование:

* Для объединения нескольких объектов в один:

var vasya = {

age: 21,

name: 'Вася',

surname: 'Петров'

};

var user = {

isAdmin: false,

isEmailConfirmed: true

};

var student = {

university: 'My university'

};

// добавить к vasya свойства из user и student

copy(vasya, user, student);

alert( vasya.isAdmin ); // false

alert( vasya.university ); // My university

* Для создания копии объекта user:

// скопирует все свойства в пустой объект

var userClone = copy({}, user);

Такой «клон» объекта может пригодиться там, где мы хотим изменять его свойства, при этом не трогая исходный объект user.

В нашей реализации мы будем копировать только свойства первого уровня, то есть вложенные объекты как-то особым образом не обрабатываются. Впрочем, её можно расширить.

А вот и реализация:

function copy() {

var dst = arguments[0];

for (var i = 1; i < arguments.length; i++) {

var arg = arguments[i];

for (var key in arg) {

dst[key] = arg[key];

}

}

return dst;

}

Здесь первый аргумент copy – это объект, в который нужно копировать, он назван dst. Для упрощения доступа к нему можно указать его прямо в объявлении функции:

function copy(dst) {

// остальные аргументы остаются безымянными

for (var i = 1; i < arguments.length; i++) {

var arg = arguments[i];

for (var key in arg) {

dst[key] = arg[key];

}

}

return dst;

}

### [Аргументы по умолчанию через ||](http://learn.javascript.ru/arguments-pseudoarray" \l "argumenty-po-umolchaniyu-cherez)

Если функция вызвана с меньшим количеством аргументов, чем указано, то отсутствующие аргументы считаются равными undefined.

Зачастую в случае отсутствия аргумента мы хотим присвоить ему некоторое «стандартное» значение или, иначе говоря, значение «по умолчанию». Это можно удобно сделать при помощи оператора логическое ИЛИ ||.

Например, функция showWarning, описанная ниже, должна показывать предупреждение. Для этого она принимает ширину width, высоту height, заголовок title и содержимое contents, но большая часть этих аргументов необязательна:

function showWarning(width, height, title, contents) {

width = width || 200; // если не указана width, то width = 200

height = height || 100; // если нет height, то height = 100

title = title || "Предупреждение";

//...

}

Это отлично работает в тех ситуациях, когда «нормальное» значение параметра в логическом контексте отлично от false. В коде выше, при передаче width = 0 или width = null, оператор ИЛИ заменит его на значение по умолчанию.

А что, если мы хотим использовать значение по умолчанию только если width === undefined? В этом случае оператор ИЛИ уже не подойдёт, нужно поставить явную проверку:

function showWarning(width, height, title, contents) {

if (width === undefined) width = 200;

if (height === undefined) height = 100;

if (title === undefined) title = "Предупреждение";

//...

}

## [Устаревшее свойство arguments.callee](http://learn.javascript.ru/arguments-pseudoarray" \l "arguments-callee)

**Используйте NFE вместо arguments.callee**

Это свойство устарело, при use strict оно не работает.

Единственная причина, по которой оно тут – это то, что его можно встретить в старом коде, поэтому о нём желательно знать.

Современная спецификация рекомендует использовать [именованные функциональные выражения (NFE)](http://learn.javascript.ru/named-function-expression#functions-nfe).

В старом стандарте JavaScript объект arguments не только хранил список аргументов, но и содержал в свойстве arguments.callee ссылку на функцию, которая выполняется в данный момент.

Например:

function f() {

alert( arguments.callee === f ); // true

}

f();

Эти два примера будут работать одинаково:

// подвызов через NFE

var factorial = function f(n) {

return n==1 ? 1 : n\*f(n-1);

};

// подвызов через arguments.callee

var factorial = function(n) {

return n==1 ? 1 : n\*arguments.callee(n-1);

};

В учебнике мы его использовать не будем, оно приведено для общего ознакомления.

### [arguments.callee.caller](http://learn.javascript.ru/arguments-pseudoarray" \l "arguments-callee-caller)

Устаревшее свойство arguments.callee.caller хранит ссылку на функцию, которая вызвала данную.

**Это свойство тоже устарело**

Это свойство было в старом стандарте, при use strict оно не работает, как и arguments.callee.

Также ранее существовало более короткое свойство arguments.caller. Но это уже раритет, оно даже не кросс-браузерное. А вот свойство arguments.callee.caller поддерживается везде, если не использован use strict, поэтому в старом коде оно встречается.

Пример работы:

f1();

function f1() {

alert( arguments.callee.caller ); // null, меня вызвали из глобального кода

f2();

}

function f2() {

alert( arguments.callee.caller ); // f1, функция, из которой меня вызвали

f3();

}

function f3() {

alert( arguments.callee.caller ); // f2, функция, из которой меня вызвали

}

В учебнике мы это свойство также не будем использовать.

## [«Именованные аргументы»](http://learn.javascript.ru/arguments-pseudoarray" \l "imenovannye-argumenty)

Именованные аргументы – альтернативная техника работы с аргументами, которая вообще не использует arguments.

Некоторые языки программирования позволяют передать параметры как-то так: f(width=100, height=200), то есть по именам, а что не передано, тех аргументов нет. Это очень удобно в тех случаях, когда аргументов много, сложно запомнить их порядок и большинство вообще не надо передавать, по умолчанию подойдёт.

Такая ситуация часто встречается в компонентах интерфейса. Например, у «меню» может быть масса настроек отображения, которые можно «подкрутить» но обычно нужно передать всего один-два главных параметра, а остальные возьмутся по умолчанию.

В JavaScript для этих целей используется передача аргументов в виде объекта, а в его свойствах мы передаём параметры.

Получается так:

function showWarning(options) {

var width = options.width || 200; // по умолчанию

var height = options.height || 100;

var contents = options.contents || "Предупреждение";

// ...

}

Вызвать такую функцию очень легко. Достаточно передать объект аргументов, указав в нем только нужные:

showWarning({

contents: "Вы вызвали функцию" // и всё понятно!

});

Сравним это с передачей аргументов через список:

showWarning(null, null, "Предупреждение!");

// мысль программиста "а что это за null, null в начале? ох, надо глядеть описание функции"

Не правда ли, объект – гораздо проще и понятнее?

Еще один бонус кроме красивой записи – возможность повторного использования объекта аргументов:

var opts = {

width: 400,

height: 200,

contents: "Текст"

};

showWarning(opts);

opts.contents = "Другой текст";

showWarning(opts); // вызвать с новым текстом, без копирования других аргументов

Именованные аргументы применяются во многих JavaScript-фреймворках.

## [Итого](http://learn.javascript.ru/arguments-pseudoarray" \l "itogo)

* Полный список аргументов, с которыми вызвана функция, доступен через arguments.
* Это псевдомассив, то есть объект, который похож на массив, в нём есть нумерованные свойства и length, но методов массива у него нет.
* В старом стандарте было свойство arguments.callee со ссылкой на текущую функцию, а также свойство arguments.callee.caller, содержащее ссылку на функцию, которая вызвала данную. Эти свойства устарели, при use strict обращение к ним приведёт к ошибке.
* Для указания аргументов по умолчанию, в тех случаях, когда они заведомо не false, удобен оператор ||.

В тех случаях, когда возможных аргументов много и, в особенности, когда большинство их имеют значения по умолчанию, вместо работы с arguments организуют передачу данных через объект, который как правило называют options.

Возможен и гибридный подход, при котором первый аргумент обязателен, а второй – options, который содержит всевозможные дополнительные параметры:

function showMessage(text, options) {

// показать сообщение text, настройки показа указаны в options

}