**Функции**

Используются, когда надо повторять одно и то же действие во многих частях программы.

Для создания функций используется объявление функции.

function имя(параметры) {

...тело...

}

одно из главных предназначений функций: избавление от дублирования кода.

## [Локальные переменные](https://learn.javascript.ru/function-basics" \l "lokalnye-peremennye)

Переменные, объявленные внутри функции, видны только внутри этой функции.

function showMessage() {

let message = "1"; // локальная переменная

}

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

Объявление Function Declaration создаёт функцию и помещает её в переменную с именем showMessage.

Функция обладает полным доступом к внешним переменным и может изменять их значение.

Внешняя переменная используется, только если внутри функции нет такой локальной.

Если одноимённая переменная объявляется внутри функции, тогда она перекрывает внешнюю.

**Глобальные переменные**

Переменные, объявленные снаружи всех функций называются *глобальными*.

*Глобальные переменные* видимы для любой функции (если только их не перекрывают одноимённые локальные переменные).

Желательно сводить использование глобальных переменных к минимуму. В современном коде обычно мало или совсем нет глобальных переменных. Хотя они иногда полезны для хранения важнейших «общепроектовых» данных.

## [Параметры](https://learn.javascript.ru/function-basics" \l "parametry)

значения переданные в параметры копируются в локальные переменные. Затем они используются в теле функции.

Если передать параметром глобальную переменную и внутри функции изменить ее, то снаружи изменений не будет видно, т.к. внутри функции изменялась копия а не сама переменная. Функция всегда получает только копию значения.

Значение, передаваемое в качестве параметра функции, также называется аргументом.

* Параметр – это переменная, указанная в круглых скобках в объявлении функции.
* Аргумент – это значение, которое передаётся функции при её вызове.

Мы объявляем функции со списком параметров, затем вызываем их, передавая аргументы.

## [Значения по умолчанию](https://learn.javascript.ru/function-basics" \l "znacheniya-po-umolchaniyu)

Если при вызове функции аргумент не был указан, то его значением становится undefined. Вызов без аргумента не приводит к ошибке.

На месте значения по умолчанию может быть и сложное выражение, которое бы вычислялось и присваивалось при отсутствии параметра.

**Использование параметров по умолчанию в ранних версиях JavaScript**

Ранние версии JavaScript не поддерживали параметры по умолчанию. Поэтому существуют альтернативные способы, которые могут встречаться в старых скриптах.

Например, явная проверка на undefined:

function showMessage(from, text) {

if (text === undefined) {

text = 'текст не добавлен';

}

alert( from + ": " + text );

}

…Или с помощью оператора ||:

function showMessage(from, text) {

// Если значение text ложно, тогда присвоить параметру text значение по умолчанию

// заметим, что при этом пустая строка text === "" будет также считаться отсутствующим значением

text = text || 'текст не добавлен';

...

}

### [Альтернативные параметры по умолчанию](https://learn.javascript.ru/function-basics" \l "alternativnye-parametry-po-umolchaniyu)

Иногда имеет смысл присваивать значения по умолчанию для параметров не в объявлении функции, а на более позднем этапе.

Во время выполнения функции мы можем проверить, передан ли параметр, сравнив его с undefined:

function showMessage(text) {

// ...

if (text === undefined) { // если параметр отсутствует

text = 'пустое сообщение';

}

alert(text);

}

showMessage(); // пустое сообщение

…Или мы можем использовать оператор ||:

function showMessage(text) {

// если значение text ложно или равняется undefined, тогда присвоить text значение 'пусто'

text = text || 'пусто';

...

}

… или [оператор нулевого слияния](https://learn.javascript.ru/nullish-operators) ??. Его использование будет лучшей практикой, в случае, если большинство ложных значений, таких как 0, следует расценивать как «нормальные».

function showCount(count) {

// если count равен undefined или null, показать "неизвестно"

alert(count ?? "неизвестно");

}

showCount(0); // 0

showCount(null); // неизвестно

showCount(); // неизвестно

## [Возврат значения](https://learn.javascript.ru/function-basics" \l "vozvrat-znacheniya)

Директива return может находиться в любом месте тела функции. Как только выполнение доходит до этого места, функция останавливается, и значение возвращается в вызвавший её код

Возможно использовать return и без значения. Это приведёт к немедленному выходу из функции.

Если функция не возвращает значения, это всё равно, как если бы она возвращала undefined Пустой return аналогичен return undefined

**Никогда не добавляйте перевод строки между return и его значением**

Перенос строки перед return воспринимается как return;

По этому, Если мы хотим, чтобы возвращаемое выражение занимало несколько строк, нужно начать его на той же строке, что и return. Или, хотя бы, поставить там открывающую скобку, вот так:

return (

some + long + expression

+ or +

whatever \* f(a) + f(b)

)

**Одна функция – одно действие**

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

Два независимых действия обычно подразумевают две функции, даже если предполагается, что они будут вызываться вместе (в этом случае мы можем создать третью функцию, которая будет их вызывать).

Имена функций, которые используются *очень часто*, иногда делают сверхкороткими.

Например, фреймворк [jQuery](http://jquery.com/) определяет функцию с помощью $. В библиотеке [Lodash](http://lodash.com/) основная функция представлена именем \_.

Это исключения. В основном имена функций должны быть в меру краткими и описательными.

Иногда разработчики называют такой код *самодокументируемым*.

Таким образом, допустимо создавать функции, даже если мы не планируем повторно использовать их. Такие функции структурируют код и делают его более понятным.