**РАСШИРЕННЫЕ ВОЗМОЖНОСТИ ФУНКЦИИ**

Аргументы

Параметры функции играют роль локальных переменных в теле функции. При указании параметров функции ключевое слово var или let использовать не нужно, JavaScript объявляет их в качестве локальных переменные автоматически.

function func (a, b, c) { ... } // a, b, c – параметры функции

Значения, которые передаются при вызове функции называются аргументами.  
При вызове функции ей передаётся список аргументов, эти аргументы присваиваются параметрам функции в том порядке, в каком они указаны: первый аргумент присваивается первому параметру, второй аргумент – второму параметру и т. д.  
Аргументы, измененные внутри функции, снаружи остаются прежними (если аргументы примитивы):

function sayHello(name) {

var prefix = "Hi Mr. ";

name = "Bart";

return prefix + name;

}

var name = "Smith";

console.log(name); // "Smith"

console.log(sayHello(name)); //"Hi Mr. Bart"

console.log(name); // "Smith"

Если внутрь функции, как аргумент, передать объект и внутри изменить его свойства, то они поменяются и снаружи.

function sayHello(param) {

var prefix = "Hi Mr. ";

param.name = "Bart";

return prefix + param.name;

}

var obj = { name: "Smith" };

console.log(obj.name); //"Smith"

console.log(sayHello(obj)); //"Hi Mr. Bart"

console.log(obj.name); //"Bart"

Это происходит потому, что примитивные типы передаются по значению, а составные типы, такие как объекты, по ссылке.

Детальнее об этом рассказано в теме про объекты.

Функцию можно вызвать с любым количеством аргументов вне зависимости от того, сколько аргументов ожидает функция.  
Если число аргументов отличается от числа параметров, то никакой ошибки не происходит.

function sayHello (firstName, secondName) {

let prefix = "Hi Mr. ";

return prefix + firstName + " " + secondName;

}

sayHello("John", "Connor"); // "Hi Mr. John Connor"

sayHello("John"); // "Hi Mr. John undefined"

Если функция имеет больше параметров, чем передано ей аргументов, то параметрам без соответствующих аргументов присваивается значение undefined.  
Есть два способа, которые помогут избежать таких ошибок.

Старый способ  
Дополнительная проверка на доступность аргументов:

function sayHello (firstName, secondName) {

let prefix = "Hi Mr. ";

return prefix + firstName + (secondName || "Nobody");

}

sayHello("John"); // "Hi Mr. John Nobody"

Новый способ (ES6)  
Параметр по умолчанию:

let sayHello = function(firstName = "Anakin", secondName = "Skywalker") {

const prefix = "Hi Mr. ";

return prefix + firstName + " " + secondName ;

}

sayHello(); // "Hi Mr. Anakin Skywalker"

sayHello("Lyke"); // "Hi Mr. Lyke Skywalker"

sayHello(null, "John"); // "Hi Mr. null John"

sayHello(undefined, undefined); // "Hi Mr. Anakin Skywalker"

Параметры функции можно инициализировать значениями по умолчанию, которые будут использоваться в том случае, если параметры не были инициализированы аргументами при вызове функции.  
Передача значения undefined в качестве аргумента воспринимается как отсутствие аргумента.

Функция, в которой используются параметры со значениями по умолчанию, всегда работает как в строгом режиме, даже если он не был включен:

function testArguments(firstName, secondName = "Skywalker") {

console.log(firstName === arguments[0]);

console.log(secondName === arguments[1]);

}

testArguments("Anakin");

// true

// false

Так как в функцию передаётся только один аргумент, arguments[1] имеет значение undefined, поэтому сравнение console.log(secondName === arguments[1]); в результате даёт false.

Если в функции используются параметры со значениями по умолчанию, то использование объявления строгого режима внутри функции приведёт к синтаксической ошибке:

function testArguments(arg1 = 1) {

"use strict"; // Ошибка

}

Значением по умолчанию может быть как простое, так и сложное выражение:

function testArguments(arg1 = 1, arg2 = 2 + 2) {

console.log(arg1, arg2);

}

testArguments(1); // 1 4

Значение предыдущего параметра можно использовать в качестве значения по умолчанию для любого из последующих параметров:

function testArguments(arg1, arg2 = arg1) {

console.log(arg1, arg2);

}

testArguments("John"); // John John

Попытка использовать значение последующего параметра в качестве значения по умолчанию для предшествующего параметра вызовет ошибку:

function testArguments(arg1 = arg2, arg2) {

console.log(arg1, arg2);

}

testArguments(undefined, "John"); // Ошибка

Параметры создаются в момент вызова функции в том порядке, в котором они следуют в определении функции. Каждый параметр создаётся по аналогии объявления переменной с помощью ключевого слова let. Это означает, что использование параметра до того как он был создан вызовет ошибку.

Псевдо-массив arguments

Arguments – локальная переменная, которая доступна внутри каждой функции.  
Она содержит информацию про количество и значение всех параметров, переданных в функцию, во время ее вызова.

function sayHello() {

console.log(arguments);

}

sayHello("John", "Connor"); // (2) ["John", "Connor"]

В функции, не принимающей аргументы на вход, все равно можно получить переданные в нее аргументы.  
Arguments отсутствует в arrow function.  
При помощи arguments можно обратиться к каждому аргументу отдельно.  
Нумерация начинается с нуля.

function sayHello(greating, name) {

console.log(arguments[0]);

console.log(arguments[1]);

console.log(arguments[2]);

console.log(greating + name + arguments[2] + arguments[3]);

}

sayHello("Hi, ", "John", " Connor", "!"); // "Hi, John Connor!"

У arguments есть единственное свойство length. С его помощю, например, можно вывести все переданные параметры при  
помощи обычного цикла for.

function sayHello(greatings, name) {

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

console.log(arguments[i]);

}

}

sayHello("Hi, ", "John", " Connor", "!");

В соответствии с typeof arguments является объектом. Но его можно превратить в настоящий массив.

function checkType() {

console.log(typeof arguments);

console.log(Array.isArray(arguments));

let arrayArguments = Array.prototype.slice.call(arguments);

console.log(Array.isArray(arrayArguments));

}

checkType();

// object

// false

// true

Детальнее о массивах мы поговорим далее в нашем курсе. В соответствующей теме.

Rest оператор

Оператор … (rest) используется вместо arguments.  
Аргументы полученные с помощью оператора … ,являются настоящим массивом, со всеми его методами и свойствами.

function rest(...args) {

console.log(args);

console.log(Array.isArray(args));

}

rest(1, 2, 3);

// (3) [1, 2, 3]

// true

Если в функцию передали больше параметров, чем она ожидает, оператор … (rest)«аккумулирует» в себе все «лишние» аргументы.

function rest(person, ...args) {

console.log(person);

console.log(...args);

}

rest("John", "Bob", "Carl");

// "John"

// "Bob Carl"

[ПРЕДЫДУЩАЯ ЧАСТЬ](https://www.jsexpert.net/course/clear-javascript-middle/?course_type=content&course_page=3&lecture=1&section-quiz)[СЛЕДУЮЩАЯ ЧАСТЬ](https://www.jsexpert.net/course/clear-javascript-middle/?course_type=section-quiz&course_page=3&lecture=2&section-quiz=1)