Set a default parameter value for a JavaScript function

Asked 10 years, 5 months ago Active today Viewed 1.2m times

2256

521

I would like a JavaScript function to have optional arguments which I set a default on, which get used if the value isn't defined (and ignored if the value is passed). In Ruby you can do it like this:

```
def read_file(file, delete_after = false)
  # code
end
```

Does this work in JavaScript?

```
function read_file(file, delete_after = false) {
  // Code
}
```

javascript function parameters default-parameters

edited May 17 at 4:29
Filipp W.
2,481 2 13 35

asked May 21 '09 at 20:07
Tilendor
23.7k 15 47 58

25 Answers

3193

From ES6/ES2015, default parameters are in the language specification.

```
function read_file(file, delete_after = false) {
  // Code
}
```

just works.

Reference: Default Parameters - MDN

> Default function parameters allow formal parameters to be initialized with default values if **no value** or **undefined** is passed.

You can also simulate default *named* parameters via destructuring:

```
// the `= {}` below lets you call the function without any parameters
function myFor({ start = 5, end = 1, step = -1 } = {}) { // (A)
    // Use the variables `start`, `end` and `step` here
    ···
}
```

Pre ES2015,

There are a lot of ways, but this is my preferred method — it lets you pass in anything you want, including false or null. (`typeof null == "object"`)

```
function foo(a, b) {
  a = typeof a !== 'undefined' ? a : 42;
  b = typeof b !== 'undefined' ? b : 'default_b';
  ...
}
```

edited May 17 at 4:38
Filipp W.
2,481 2 13 35

answered May 21 '09 at 20:10
Tom Ritter
85.7k 28 127 163

207 You can also encapsulate it as such: `function defaultFor(arg, val) { return typeof arg !== 'undefined' ? arg : val; }` and then you can call it as `a = defaultFor(a, 42);` – Camilo Martin Sep 2 '12 at 5:56

5 @SiPlus and you got extra reference errors for free of charge while trying to use `undefined` objects :p Even while it may work with some browsers and might be faster, `null` is still an object and `undefined` is reference to primitive type that is trying to tell that there is nothing here, not even `null` . See here, both cases in nutshell: JavaScript/Reference/Global_Objects/undefined. – Sampo Sarrala Nov 3 '12 at 1:29

7 if you check against the property of an object, then `typeof` is redundant. `function foo(data) { var bar = data.bar !== undefined ? data.bar : 'default'; }` This won't throw reference errors and is concise. – Dziamid May 17 '13 at 11:06

8 I know it's really minor, but I don't like how you are assigning `a = a` and `b = b` when those parameters are not undefined. Also, that ternary operator with a bit complicated condition may be hard to read for future maintainers. I would prefer following syntax: `if (typeof a == 'undefined') a = 42` - no unnecessary assignment plus a little bit easier to read. – Daddy32 Dec 9 '14 at 15:31

9 Is there any reason to use `typeof` ? It would seem simpler to just do `a === undefined` . Plus that way you'd get a reference error if you misspelled `undefined` vs. putting it in a string. – Abe Voelker May 21 '15 at 22:04

590

```
function read_file(file, delete_after) {
    delete_after = delete_after || "my default here";
    //rest of code
}
```

This assigns to `delete_after` the value of `delete_after` if it is not a *falsey* value otherwise it assigns the string `"my default here"` . For more detail, check out Doug Crockford's survey of the language and check out the section on Operators.

This approach does not work if you want to pass in a *falsey* value i.e. `false` , `null` , `undefined` , `0` or `""` . If you require *falsey* values to be passed in you would need to use the method in Tom Ritter's answer.

When dealing with a number of parameters to a function, it is often useful to allow the consumer to pass the parameter arguments in an object and then *merge* these values with an object that contains the default values for the function

```
function read_file(values) {
    values = merge({
        delete_after : "my default here"
    }, values || {});

    // rest of code
}

// simple implementation based on $.extend() from jQuery
function merge() {
    var obj, name, copy,
        target = arguments[0] || {},
        i = 1,
        length = arguments.length;

    for (; i < length; i++) {
        if ((obj = arguments[i]) != null) {
            for (name in obj) {
                copy = obj[name];

                if (target === copy) {
                    continue;
                }
                else if (copy !== undefined) {
                    target[name] = copy;
                }
            }
        }
    }

    return target;
};
```

to use

```
// will use the default delete_after value
read_file({ file: "my file" });

// will override default delete_after value
read_file({ file: "my file", delete_after: "my value" });
```

edited Jan 9 '18 at 10:30
Vojtech Ruzicka
10.3k 15 46 51

answered May 21 '09 at 20:09
Russ Cam
108k 24 173 235

112 I find this insufficient, because I may want to pass in false. – Tom Ritter May 21 '09 at 20:11

52 I find it's adequate for most situations – Russ Cam May 21 '09 at 20:16

38 This also doesn't work for 0. :(– Alex Kahn Jun 13 '12 at 15:07

44 It doesn't work for any falsey value – Russ Cam Jun 18 '12 at 6:20

49 Because it doesn't work for falsey values, it may create a maintenance nightmare. A bit of code that has always been passed truthy values before and suddenly fails because a falsey one is passed in should probably be avoided where a more robust approach is available. – jinglesthula Oct 31 '12 at 20:34

144

I find something simple like this to be much more concise and readable personally.

```
function pick(arg, def) {
   return (typeof arg == 'undefined' ? def : arg);
}

function myFunc(x) {
  x = pick(x, 'my default');
}
```

answered May 21 '09 at 20:18
tj111
18.8k 6 55 75

6 Update: If you're using **underscore.js** already I find it even better to use `_.defaults(iceCream, {flavor: "vanilla", sprinkles: "lots"});` . Using the global namespace as shown in this answer is by many considered a bad practice. You may also consider rolling your own utility for this common task (eg. `util.default(arg, "defaul value")`) if don't want to use underscore, but I mostly end up using underscore sooner or later anyway no point in reinventing the wheel. – andersand Aug 15 '14 at 20:12

2 I actually recommend this one, i use it and call it "por" which stands for "parameter or" – super Apr 21 '15 at 16:59

2 Don't say typeof arg == 'undefined', instead say arg === undefined – OsamaBinLogin Oct 20 '15 at 15:06

3 @OsamaBinLogin what you say is correct for current JS - the older test persists because on some browsers it used to be possible to overwrite `undefined` with some other value, causing the test to fail. – Alnitak Nov 16 '15 at 11:34

2 @Alnitak: It's still possible to override `undefined` . So, it's still a good idea to use `typeof` and `'undefined'` . – L S Jun 14 '17 at 20:27

62

In ECMAScript 6 you will actually be able to write exactly what you have:

```
function read_file(file, delete_after = false) {
  // Code
}
```

This will set `delete_after` to `false` if it s not present or `undefined` . You can use ES6 features like this one today with transpilers such as Babel.

See the MDN article for more information.

edited Jun 1 '15 at 20:43

answered May 29 '15 at 15:25
Felix Kling
595k 137 911 974

1 ECMAScript 6 I suppose... (I would have corrected by myself but I can't make edits < 6 chars) – Zac Jun 1 '15 at 20:15

1 Waiting for browser any get ECMAScript 6 – Adriano Resende Jul 8 '15 at 13:05

1 What would Babel transpile this to? – harryg Jul 9 '15 at 11:31

2 @harryg: babeljs.io/repl/… – Felix Kling Jul 9 '15 at 13:22

2 Here is the compatibility table for ES6 kangax.github.io/compat-table/es6/#default_function_parameters Unfortunately this syntax **isn't supported yet**. – freemanoid Jul 9 '15 at 19:03

26

Default Parameter Values

With ES6, you can do perhaps one of the most common idioms in `JavaScript` relates to setting a default value for a function parameter. The way we've done this for years should look quite familiar:

```
function foo(x,y) {
 x = x || 11;
 y = y || 31;
 console.log( x + y );
}
foo(); // 42
foo( 5, 6 ); // 11
foo( 5 ); // 36
foo( null, 6 ); // 17
```

This pattern is most used, but is dangerous when we pass values like

```
foo(0, 42)
foo( 0, 42 ); // 53 <-- Oops, not 42
```

Why? Because the `0 is falsy`, and so the `x || 11 results in 11`, not the directly passed in 0. To fix this gotcha, some people will instead write the check more verbosely like this:

```
function foo(x,y) {
 x = (x !== undefined) ? x : 11;
 y = (y !== undefined) ? y : 31;
 console.log( x + y );
}
foo( 0, 42 ); // 42
foo( undefined, 6 ); // 17
```

we can now examine a nice helpful syntax added as of `ES6` to streamline the assignment of default values to missing arguments:

```
function foo(x = 11, y = 31) {
 console.log( x + y );
}

foo(); // 42
foo( 5, 6 ); // 11
foo( 0, 42 ); // 42
foo( 5 ); // 36
foo( 5, undefined ); // 36 <-- `undefined` is missing
foo( 5, null ); // 5 <-- null coerces to `0`
foo( undefined, 6 ); // 17 <-- `undefined` is missing
foo( null, 6 ); // 6 <-- null coerces to `0`
```

`x = 11` in a function declaration is more like `x !== undefined ? x : 11` than the much more common idiom `x || 11`

Default Value Expressions

`Function` default values can be more than just simple values like 31; they can be any valid expression, even a `function call` :

```
function bar(val) {
 console.log( "bar called!" );
 return y + val;
}
function foo(x = y + 3, z = bar( x )) {
 console.log( x, z );
}
var y = 5;
foo(); // "bar called"
 // 8 13
foo( 10 ); // "bar called"
 // 10 15
y = 6;
foo( undefined, 10 ); // 9 10
```

As you can see, the default value expressions are lazily evaluated, meaning they're only run if and when they're needed — that is, when a parameter's argument is omitted or is undefined.

A default value expression can even be an inline function expression call — commonly referred to as an Immediately Invoked Function Expression `(IIFE)` :

```
function foo( x =
 (function(v){ return v + 11; })( 31 )
) {
 console.log( x );
}
foo(); // 42
```

answered Oct 15 '16 at 17:58

Thalaivar
17.1k 4 52 65

11

that solution is work for me in js:

```
function read_file(file, delete_after) {
    delete_after = delete_after || false;
    // Code
}
```

answered Nov 16 '15 at 11:29

Takács Zsolt
180 3 13

4 What happens if `delete_after` is `0` or `null` ? It will not work correct for these cases. – Dmitri Pavlutin Dec 20 '15 at 9:33

@StephanBijzitter it only would be true in some cases. But if you only want the default values to unset values this didn't work, because 0 or null !=== false. – Rutrus Feb 1 '17 at 10:13

@Rutrus: I don't understand anything you just said. – Stephan Bijzitter Feb 1 '17 at 10:21

What about `delete_after = delete_after === undefined ? false : delete_after` ? – aashah7 Apr 28 '17 at 3:50

9

Just use an explicit comparison with undefined.

```
function read_file(file, delete_after)
{
    if(delete_after === undefined) { delete_after = false; }
}
```

answered Nov 16 '15 at 7:34

Martin Wantke
2,133 18 13

8

As an update...with ECMAScript 6 you can **FINALLY** set default values in function parameter declarations like so:

```
function f (x, y = 7, z = 42) {
  return x + y + z
}

f(1) === 50
```

As referenced by - http://es6-features.org/#DefaultParameterValues

answered Oct 6 '15 at 16:16
zillaofthegods
1,218 3 18 46

10 This answer is not useful because it is a duplicate – user Nov 6 '15 at 0:48

7

being a long time C++ developer (Rookie to web development :)), when I first came across this situation, I did the parameter assignment in the function definition, like it is mentioned in the question, as follows.

```
function myfunc(a,b=10)
```

But beware that it doesn't work consistently across browsers. For me it worked on chrome on my desktop, but did not work on chrome on android. Safer option, as many have mentioned above is -

```
    function myfunc(a,b)
    {
    if (typeof(b)==='undefined') b = 10;
......
    }
```

Intention for this answer is not to repeat the same solutions, what others have already mentioned, but to inform that parameter assignment in the function definition may work on some browsers, but don't rely on it.

answered May 19 '16 at 6:40
vivek
207 3 8

3 Assignment within the function definition also doesn't work in IE 11, which is the most current version of IE for Windows 8.1. – DiMono Jul 19 '16 at 2:52

1 In case anyone's wondering, `function myfunc(a,b=10)` is ES6 syntax, there are links to compatibility tables in other answers. – gcampbell Sep 18 '16 at 9:00

7

I would highly recommend extreme caution when using default parameter values in javascript. It often creates bugs when used in conjunction with higher order functions like `forEach`, `map`, and `reduce`. For example, consider this line of code:

```
['1', '2', '3'].map(parseInt); // [1, NaN, NaN]
```

parseInt has an optional second parameter `function parseInt(s, [` **radix** `=10])` but map calls `parseInt` with three arguments: (*element*, *index*, and *array*).

I suggest you separate your required parameters form your optional/default valued arguments. If your function takes 1,2, or 3 required parameters for which no default value makes sense, make them positional parameters to the function, any optional parameters should follow as named attributes of a single object. If your function takes 4 or more, perhaps it makes more sense to supply all arguments via attributes of a single object parameter.

In your case I would suggest you write your deleteFile function like this: (*edited per* `instead` *'s comments*)...

```
// unsafe
function read_file(fileName, deleteAfter=false) {
    if (deleteAfter) {
        console.log(`Reading and then deleting ${fileName}`);
    } else {
        console.log(`Just reading ${fileName}`);
    }
}

// better
function readFile(fileName, options) {
  const deleteAfter = !!(options && options.deleteAfter === true);
  read_file(fileName, deleteAfter);
}
```

```
console.log('unsafe...');
['log1.txt', 'log2.txt', 'log3.txt'].map(read_file);

console.log('better...');
['log1.txt', 'log2.txt', 'log3.txt'].map(readFile);
```

Run code snippet Expand snippet

Running the above snippet illustrates the dangers lurking behind default argument values for unused parameters.

edited Jan 28 at 1:59

answered Oct 15 '17 at 22:36
Doug Coburn
1,469 12 17

This "better solution" is quite unreadable. I would rather recommend to check variable type like `typeof deleteAfter === 'bool'` and throw Exception otherwise. Plus in case of using methods like map/foreach etc. just use them with caution and wrap called functions with anonymous function. `['1', '2', '3'].map((value) => {read_file(value);})` – instead Jan 26 at 17:36

That's a good point. I was trying to avoid using libraries in my code snippet, but this idiom is quite messy in pure javascript. Personally, I would use lodash's `get` method to retrieve deleteAfter. Throwing an exception if `typeof 'deleteAfter' !== 'bool'` may also be a good prudent measure. I don't like designing methods that require the caller to "use caution". Caution is the responsibility of the function, not the caller. The end behavior should follow the principle of least surprise. – Doug Coburn Jan 26 at 18:18

I agree, that method shouldn't be wrote the way, that calling it, can broke its behavior. But keep in mind that function is like model, and caller is like controller. Model should take care of data processing. Function (method) should expect that data, can be passed wrong way and handle it. That's why checking type of parameter plus throwing Exception, is the best way. Even though it's more complex. But still it can save You from scratching Your head and keep asking... WHY IT IS NOT WORKING?! and then... WHY IT IS WORKING?! – instead Jan 26 at 23:52

6

To anyone interested in having there code work in Microsoft Edge, do not use defaults in function parameters.

```
function read_file(file, delete_after = false) {
    #code
}
```

In that example Edge will throw an error "Expecting ')'"

To get around this use

```
function read_file(file, delete_after) {
  if(delete_after == undefined)
  {
    delete_after = false;
  }
  #code
}
```

As of Aug 08 2016 this is still an issue

edited Sep 23 '16 at 20:15

answered Aug 3 '16 at 14:49

Steven Johnston
1,106 10 16

3

As per the syntax

```
function [name]([param1[ = defaultValue1 ][, ..., paramN[ = defaultValueN ]]]) {
   statements
}
```

you can define the default value of formal parameters. and also check undefined value by using **typeof** function.

answered Oct 15 '17 at 20:40

sachin kumar
531 1 4 18

3

ES6: As already mentioned in most answers, in ES6, you can simply initialise a parameter along with a value.

ES5: Most of the given answers aren't good enough for me because there are occasions where I may have to pass falsey values such as `0`, `null` and `undefined` to a function. To determine if a parameter is undefined because that's the value I passed instead of undefined due to not have been defined at all I do this:

```
function foo (param1, param2) {
   param1 = arguments.length >= 1 ? param1 : "default1";
```

```
    param2 = arguments.length >= 2 ? param2 : "default2";
}
```

answered Aug 12 '18 at 5:26

Angel Politis
8,089 12 35 55

3

```
function helloWorld(name, symbol = '!!!') {
    name = name || 'worlds';
    console.log('hello ' + name + symbol);
}

helloWorld(); // hello worlds!!!

helloWorld('john'); // hello john!!!

helloWorld('john', '(>.<)'); // hello john(>.<)

helloWorld('john', undefined); // hello john!!!

helloWorld(undefined, undefined); // hello worlds!!!
```

answered Jun 8 '18 at 8:55

Tính Ngô Quang
1,836 15 17

3

Use this if you want to use latest `ECMA6` syntax:

```
function myFunction(someValue = "This is DEFAULT!") {
  console.log("someValue --> ", someValue);
}

myFunction("Not A default value") // calling the function without default value
myFunction()  // calling the function with default value
```

Run code snippet   Expand snippet

It is called `default function parameters` . It allows formal parameters to be initialized with default values if no value or undefined is passed. **NOTE**: It wont work with Internet Explorer or older browsers.

For maximum possible **compatibility** use this:

```
function myFunction(someValue) {
  someValue = (someValue === undefined) ? "This is DEFAULT!" : someValue;
  console.log("someValue --> ", someValue);
}

myFunction("Not A default value") // calling the function without default value
myFunction()  // calling the function with default value
```

Run code snippet   Expand snippet

Both functions have exact same behavior as each of these example rely on the fact that the parameter variable will be `undefined` if no parameter value was passed when calling that function.

answered Jul 5 '18 at 11:54
BlackBeard
6,663 5 33 45

NOTE: as @BlackBeard says, function(param=value) WONT work with IE. Use the compatible version. – MagicLAMP Nov 16 '18 at 2:31

2

If you are using `ES6+` you can set default parameters in the following manner:

```
function test (foo = 1, bar = 2) {
  console.log(foo, bar);
}

test(5); // foo gets overwritten, bar remains default parameter
```

Run code snippet   Expand snippet

If you need `ES5` syntax you can do it in the following manner:

```
function test(foo, bar) {
  foo = foo || 2;
  bar = bar || 0;

  console.log(foo, bar);
}

test(5); // foo gets overwritten, bar remains default parameter
```

Run code snippet   Expand snippet

In the above syntax the `OR` operator is used. The `OR` operator always returns the first value if this can be converted to `true` if not it returns the righthandside value. When the function is called with no corresponding argument the parameter variable (`bar` in our example) is set to `undefined` by the JS engine. `undefined` Is then converted to false and thus does the `OR` operator return the value 0.

answered Aug 8 '18 at 15:48
Willem van der Veen
8,249 4 59 57

2

```
function throwIfNoValue() {
throw new Error('Missing argument');
}
function foo(argValue = throwIfNoValue()) {
return argValue ;
}
```

Run code snippet   Expand snippet

Here foo() is a function which has a parameter named argValue. If we don't pass anything in the function call here, then the function throwIfNoValue() will be called and the returned result will be assigned to the only argument argValue. This is how a function call can be used as a default parameter. Which makes the code more simplified and readable.

This example has been taken from here

answered Jul 1 '18 at 8:50
dutta
41 2

1

If for some reason you are **not** on ES6 and **are** using `lodash` here is a concise way to default function parameters via `_.defaultTo` method:

```
var fn = function(a, b) {
  a = _.defaultTo(a, 'Hi')
  b = _.defaultTo(b, 'Mom!')

  console.log(a, b)
}

fn()                 // Hi Mom!
fn(undefined, null)  // Hi Mom!
fn(NaN, NaN)         // Hi Mom!
fn(1)                // 1 "Mom!"
fn(null, 2)          // Hi 2
fn(false, false)     // false false
fn(0, 2)             // 0 2


<script src="https://cdnjs.cloudflare.com/ajax/libs/lodash.js/4.17.11/lodash.min.js">
</script>
```

Run code snippet Expand snippet

Which will set the default if the current value is **NaN**, **null**, or **undefined**

edited Oct 10 '18 at 7:10 answered Oct 10 '18 at 7:04
Akrion
13.3k 1 18 31

1

Yes, using default parameters is fully supported in **ES6**:

```
function read_file(file, delete_after = false) {
  // Code
}
```

or

```
const read_file = (file, delete_after = false) => {
    // Code
}
```

but prior in **ES5** you could easily do this:

```
function read_file(file, delete_after) {
  var df = delete_after || false;
  // Code
}
```

Which means if the value is there, use the value, otherwise, use the second value after `||` operation which does the same thing...

Note: also there is a big difference between those if you pass a value to **ES6** one even the value be falsy, that will be replaced with new value, something like `null` or `""` ... but **ES5** one only will be replaced if only the passed value is truthy, that's because the way `||` working...

edited Jan 25 at 8:15

answered Jan 18 at 7:29

Alireza

62.9k 15 202 136

0

```
def read_file(file, delete_after = false)
  # code
end
```

Following code may work in this situation including ECMAScript 6 (ES6) as well as earlier versions.

```
function read_file(file, delete_after) {
    if(delete_after == undefined)
        delete_after = false;//default value

    console.log('delete_after =',delete_after);
}
read_file('text1.txt',true);
read_file('text2.txt');
```

Run code snippet   Expand snippet

as default value in languages works when the function's parameter value is skipped when calling, in JavaScript it is assigned to **undefined**. This approach doesn't look attractive programmatically but have **backward compatibility**.

answered Sep 24 '18 at 23:02
Muhammad Rizwan
183 1 12

0

Sounds of Future

In future, you will be able to "spread" one object to another (currently as of 2019 NOT supported by Edge!) - demonstration how to use that for nice default options regardless of order:

```
function test(options) {
    var options = {
       // defaults
       url: 'defaultURL',
       some: 'somethingDefault',
       // override with input options
       ...options
    };

    var body = document.getElementsByTagName('body')[0];
    body.innerHTML += '<br>' + options.url + ' : ' + options.some;
}
test();
test({});
test({url:'myURL'});
test({some:'somethingOfMine'});
test({url:'overrideURL', some:'andSomething'});
test({url:'overrideURL', some:'andSomething', extra:'noProblem'});
```

Run code snippet Expand snippet

MDN reference: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Spread_syntax

...meanwhile what Edge DOES support is Object.assign() (IE does not, but I really hope we can leave IE behind :))

Similarly you could do

```
    function test(options) {
        var options = Object.assign({
           // defaults
           url: 'defaultURL',
           some: 'somethingDefault',
        }, options); // override with input options

        var body = document.getElementsByTagName('body')[0];
        body.innerHTML += '<br>' + options.url + ' : ' + options.some;
    }
    test();
    test({});
    test({url:'myURL'});
    test({some:'somethingOfMine'});
    test({url:'overrideURL', some:'andSomething'});
    test({url:'overrideURL', some:'andSomething', extra:'noProblem'});
```

Run code snippet Expand snippet

answered 6 hours ago
jave.web
7,602 9 56 83

Yeah this is referred to as a default parameter

Default function parameters allow formal parameters to be initialized with default values if no value or undefined is passed.

0

Syntax:

```
function [name]([param1[ = defaultValue1 ][, ..., paramN[ = defaultValueN ]]]) {
   statements
}
```

Description:

Parameters of functions default to undefined However, in situations it might be useful to set a different default value. This is where default parameters can help.

In the past, the general strategy for setting defaults was to test parameter values in the body of the function and assign a value if they are undefined. If no value is provided in the call, its value would be undefined. You would have to set a conditional check to make sure the parameter is not undefined

With default parameters in ES2015, the check in the function body is no longer necessary. Now you can simply put a default value in the function head.

Example of the differences:

```
// OLD METHOD
function multiply(a, b) {
  b = (typeof b !== 'undefined') ?  b : 1;
  return a * b;
}

multiply(5, 2); // 10
multiply(5, 1); // 5
multiply(5);    // 5


// NEW METHOD
function multiply(a, b = 1) {
  return a * b;
}

multiply(5, 2); // 10
multiply(5, 1); // 5
multiply(5);    // 5
```

Different Syntax Examples:

Padding undefined vs other falsy values:

Even if the value is set explicitly when calling, the value of the num argument is the default one.

```
function test(num = 1) {
  console.log(typeof num);
}

test();          // 'number' (num is set to 1)
test(undefined); // 'number' (num is set to 1 too)

// test with other falsy values:
test('');        // 'string' (num is set to '')
test(null);      // 'object' (num is set to null)
```

Evaluated at call time:

The default argument gets evaluated at call time, so unlike some other languages, a new object is created each time the function is called.

```
function append(value, array = []) {
  array.push(value);
  return array;
}

append(1); //[1]
append(2); //[2], not [1, 2]


// This even applies to functions and variables
function callSomething(thing = something()) {
 return thing;
}

function something() {
  return 'sth';
}

callSomething();  //sth
```

Default parameters are available to later default parameters:

Params already encountered are available to later default parameters

```
function singularAutoPlural(singular, plural = singular + 's',
                        rallyingCry = plural + ' ATTACK!!!') {
```

```
  return [singular, plural, rallyingCry];
}

//["Gecko","Geckos", "Geckos ATTACK!!!"]
singularAutoPlural('Gecko');

//["Fox","Foxes", "Foxes ATTACK!!!"]
singularAutoPlural('Fox', 'Foxes');

//["Deer", "Deer", "Deer ... change."]
singularAutoPlural('Deer', 'Deer', 'Deer peaceably and respectfully \ petition the
government for positive change.')
```

Functions defined inside function body:

Introduced in Gecko 33 (Firefox 33 / Thunderbird 33 / SeaMonkey 2.30). Functions declared in the function body cannot be referred inside default parameters and throw a ReferenceError (currently a TypeError in SpiderMonkey, see bug 1022967). Default parameters are always executed first, function declarations inside the function body evaluate afterwards.

```
// Doesn't work! Throws ReferenceError.
function f(a = go()) {
  function go() { return ':P'; }
}
```

Parameters without defaults after default parameters:

Prior to Gecko 26 (Firefox 26 / Thunderbird 26 / SeaMonkey 2.23 / Firefox OS 1.2), the following code resulted in a SyntaxError. This has been fixed in bug 777060 and works as expected in later versions. Parameters are still set left-to-right, overwriting default parameters even if there are later parameters without defaults.

```
function f(x = 1, y) {
  return [x, y];
}

f(); // [1, undefined]
f(2); // [2, undefined]
```

Destructured paramet with default value assignment:

You can use default value assignment with the destructuring assignment notation

```
function f([x, y] = [1, 2], {z: z} = {z: 3}) {
  return x + y + z;
}

f(); // 6
```

answered Sep 27 '18 at 16:56
esewalson
102 2

0

Just a different approach to set default params is to use object map of arguments, instead of arguments directly. For example,

```
const defaultConfig = {
 category: 'Animals',
 legs: 4
};

function checkOrganism(props) {
 const category = props.category || defaultConfig.category;
 const legs = props.legs || defaultConfig.legs;
}
```

This way, it's easy to extend the arguments and not worry about argument length mismatch.

answered Jun 13 at 9:26
Aman Pandey
126 1 1 12

0

The answer is yes. In fact, there are many languages who support default parameters. Python is one of them:

```
def(a, enter="Hello"):
   print(a+enter)
```

Even though this is Python 3 code due to the parentheses, default parameters in functions also work in JS.

For example, and in your case:

```
function read_file(file, deleteAfter=false){
  console.log(deleteAfter);
}

read_file("test.txt");
```

Run code snippet Expand snippet

But sometimes you don't really need default parameters.

You can just define the variable right after the start of the function, like this:

```
function read_file(file){
  var deleteAfter = false;
  console.log(deleteAfter);
}

read_file("test.txt");
```

Run code snippet Expand snippet

In both of my examples, it returns the same thing. But sometimes they actually could be useful, like in very advanced projects.

So, in conclusion, default parameter values can be used in JS. But it is almost the same thing as defining a variable right after the start of the function. However, sometimes they are still very useful. As you have may noticed, default parameter values take 1 less line of code than the standard way which is defining the parameter right after the start if the function.

EDIT: And this is super important! This will *not* work in IE. See documentation. So with IE you have to use the "define variable at top of function" method. Default parameters won't work in IE.

edited Sep 1 at 17:16

answered Jul 16 at 16:26

zixuan
482 5 15

Yes, This will work in Javascript. You can also do that:

0

```
function func(a=10,b=20)
{
    alert (a+' and '+b);
}

func(); // Result: 10 and 20

func(12); // Result: 12 and 20

func(22,25); // Result: 22 and 25
```

answered Jul 19 '16 at 11:59

Muhammad Awais
2,402 1 29 26

2 It is not currently portable... IE... developer.mozilla.org/en/docs/Web/JavaScript/Reference/… (And it is a duplicate answer...) – Gert van den Berg May 11 '17 at 8:34

protected by Samuel Liew ♦ Oct 5 '15 at 9:18

Thank you for your interest in this question. Because it has attracted low-quality or spam answers that had to be removed, posting an answer now requires 10 reputation on this site (the association bonus does not count).

Would you like to answer one of these unanswered questions instead?