JavaScript Functions

**Summary**: in this tutorial, you will learn about JavaScript functions and how to use them to structure the code into smaller and more reusable units.

Introduction to JavaScript functions

When developing an application, you often need to perform the same action in many places. For example, you may want to show a message whenever an error occurs.

To avoid repeating the same code all over places, you can use a function to wrap that code and reuse it.

JavaScript provides many built-in functions such as parseInt() and parseFloat(). In this tutorial, you will learn how to develop custom functions.

Declare a function

To declare a function, you use the function keyword, followed by the function name, a list of parameters, and the function body as follows:

function functionName(parameters) {

*// function body*

*// ...*

}

Code language: JavaScript (javascript)

The function name must be a valid JavaScript identifier. By convention, the function names are in camelCase and start with verbs like getData(), fetchContents(), and isValid().

A function can accept zero, one, or multiple parameters. In the case of multiple parameters, you need to use a comma to separate two parameters.

The following declares a function say() that accepts no parameter:

function say() {

}

Code language: JavaScript (javascript)

The following declares a function named square() that accepts one parameter:

function square(a) {

}

Code language: JavaScript (javascript)

And the following declares a function named add() that accepts two parameters:

function add(a, b) {

}

Code language: JavaScript (javascript)

Inside the function body, you can write the code to implement an action. For example, the following say() function simply shows a message to the console:

function say(message) {

console.log(message);

}

Code language: JavaScript (javascript)

In the body of the say() function, we call the console.log() function to output a message to the console.

Calling a function

To use a function, you need to call it. Calling a function is also known as invoking a function. To call a function, you use its name followed by arguments enclosing in parentheses like this:

functionName(arguments);

Code language: JavaScript (javascript)

When calling a function, JavaScript executes the code inside the function body. For example, the following shows how to call the say() function:

say('Hello');

Code language: JavaScript (javascript)

In this example, we call the say() function and pass a literal string 'Hello' into it.

Parameters vs. Arguments

The terms parameters and arguments are often used interchangeably. However, they are essentially different.

When declaring a function, you specify the parameters. However, when calling a function, you pass the arguments that are corresponding to the parameters.

For example, in the say() function, the message is the parameter and the 'Hello' string is an argument that corresponds to the message parameter.

Returning a value

Every function in JavaScript implicitly returns undefined unless you explicitly specify a return value. For example:

function say(message) {

console.log(message);

}

let result = say('Hello');

console.log('Result:', result);

Code language: JavaScript (javascript)

Output:

Hello

Result: undefined

Code language: JavaScript (javascript)

To specify a return value for a function, you use the return statement followed by an expression or a value, like this:

return expression;

Code language: JavaScript (javascript)

For example, the following add() function returns the sum of the two arguments:

function add(a, b) {

return a + b;

}

Code language: JavaScript (javascript)

The following shows how to call the add() function:

let sum = add(10, 20);

console.log('Sum:', sum);

Code language: JavaScript (javascript)

Output:

Sum: 30

Code language: HTTP (http)

The following example uses multiple return statements in a function to return different values based on conditions:

function compare(a, b) {

if (a > b) {

return -1;

} else if (a < b) {

return 1;

}

return 0;

}

Code language: JavaScript (javascript)

The compare() function compares two values. It returns:

* -1 if the first argument is greater than the second one.
* 1 if the first argument is less than the second one.
* 0 if  the first argument equals the second one.

The function immediately stops executing immediately when it reaches the return statement. Therefore, you can use the return statement without a value to exit the function prematurely, like this:

function say(message) {

*// show nothing if the message is empty*

if (! message ) {

return;

}

console.log(message);

}

Code language: JavaScript (javascript)

In this example, if the message is blank (or undefined), the say() function will show nothing.

The function can return a single value. If you want to [return multiple values from a function](https://www.javascripttutorial.net/javascript-return-multiple-values/), you need to pack these values in an array or an object.

The arguments object

Inside a function, you can access an object called arguments that represents the named arguments of the function.

The arguments object behaves like an [array](https://www.javascripttutorial.net/javascript-array/) though it is not an instance of the [Array](https://www.javascripttutorial.net/javascript-array/) type.

For example, you can use the square bracket [] to access the arguments: arguments[0] returns the first argument, arguments[1] returns the second one, and so on.

Also, you can use the length property of the arguments object to determine the number of arguments.

The following example implements a generic add() function that calculates the sum of any number of arguments.

function add() {

let sum = 0;

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

sum += arguments[i];

}

return sum;

}

Code language: JavaScript (javascript)

Hence, you can pass any number of arguments to the add() function, like this:

console.log(add(1, 2)); *// 3*

console.log(add(1, 2, 3, 4, 5)); *// 15*

Code language: JavaScript (javascript)

Function hoisting

In JavaScript, you can use a function before declaring it. For example:

showMe(); *// a hoisting example*

function showMe(){

console.log('an hoisting example');

}

Code language: JavaScript (javascript)

This feature is called [hoisting](https://www.javascripttutorial.net/javascript-hoisting/).

Function hoisting is a mechanism that the JavaScript engine physically moves function declarations to the top of the code before executing them.

The following shows the version of the code before the JavaScript engine executes it:

function showMe(){

console.log('a hoisting example');

}

showMe(); *// a hoisting example*

Code language: JavaScript (javascript)

Summary

* Use the function keyword to declare a function.
* Use the functionName() to call a function.
* All functions implicitly return undefined if they don’t explicitly return a value.
* Use the return statement to return a value from a function explicitly.
* The arguments variable is an array-like object inside a function, representing function arguments.
* The function hoisting allows you to call a function before declaring it.

JavaScript Functions are First-Class Citizens

**Summary**: in this tutorial, you’ll learn that JavaScript functions are first-class citizens. This means that you can store functions in variables, pass them to other functions as arguments, and return them from other functions as values.

Storing functions in variables

[Functions](https://www.javascripttutorial.net/javascript-function/) are first-class citizens in JavaScript. In other words, you can treat functions like values of other types.

The following defines the add() function and assigns the function name to the variable sum:

function add(a, b) {

return a + b;

}

let sum = add;

Code language: JavaScript (javascript)

In the assignment statement, we don’t include the opening and closing parentheses at the end of the add identifier. We also don’t execute the function but reference the function.

By doing this, we can have two ways to execute the same function. For example, we can call it normally as follows:

let result = add(10, 20);

Code language: JavaScript (javascript)

Alternatively, we can all the add() function via the sum variable like this:

let result = sum(10,20);

Code language: JavaScript (javascript)

Passing a function to another function

Because functions are values, you can pass a function as an argument into another function.

The following declares the average() function that takes three arguments. The third argument is a function:

function average(a, b, fn) {

return fn(a, b) / 2;

}

Code language: JavaScript (javascript)

Now, you can pass the sum function to the average() function as follows:

let result = average(10, 20, sum);

Code language: JavaScript (javascript)

Put it all together:

function add(a, b) {

return a + b;

}

let sum = add;

function average(a, b, fn) {

return fn(a, b) / 2;

}

let result = average(10, 20, sum);

console.log(result);

Code language: JavaScript (javascript)

Output:

15

Returning functions from functions

Since functions are values, you can return a function from another function.

The following compareBy() function returns a function that compares two [objects](https://www.javascripttutorial.net/javascript-objects/) by a property:

function compareBy(propertyName) {

return function (a, b) {

let x = a[propertyName],

y = b[propertyName];

if (x > y) {

return 1;

} else if (x < y) {

return -1;

} else {

return 0;

}

};

}

Code language: JavaScript (javascript)

Note that a[propertyName] returns the value of the propertyName of the a object. It’s equivalent to a.propertyName. However, if the propertyName contains a space like 'Discount Price', you need to use the square bracket notation to access it.

Suppose that you have an array of product objects where each product object has two properties: name and price.

let products = [

{name: 'iPhone', price: 900},

{name: 'Samsung Galaxy', price: 850},

{name: 'Sony Xperia', price: 700}

];

Code language: JavaScript (javascript)

You can [sort an array](https://www.javascripttutorial.net/javascript-array-sort/) by calling the sort() method. The sort() method accepts a function that compares two elements of the array as an argument.

For example, you can sort the product objects based on the name by passing a function returned from the compareBy() function as follows:

console.log('Products sorted by name:');

products.sort(compareBy('name'));

console.table(products);

Code language: JavaScript (javascript)

Output:

Products sorted by name:

┌─────────┬──────────────────┬───────┐

│ (index) │ name │ price │

├─────────┼──────────────────┼───────┤

│ 0 │ 'Samsung Galaxy' │ 850 │

│ 1 │ 'Sony Xperia' │ 700 │

│ 2 │ 'iPhone' │ 900 │

└─────────┴──────────────────┴───────┘

Code language: plaintext (plaintext)

Similarly, you can sort the product objects by prices:

*// sort products by prices*

console.log('Products sorted by price:');

products.sort(compareBy('price'));

console.table(products);

Code language: JavaScript (javascript)

Output:

Products sorted by price:

┌─────────┬──────────────────┬───────┐

│ (index) │ name │ price │

├─────────┼──────────────────┼───────┤

│ 0 │ 'Sony Xperia' │ 700 │

│ 1 │ 'Samsung Galaxy' │ 850 │

│ 2 │ 'iPhone' │ 900 │

└─────────┴──────────────────┴───────┘

Code language: JavaScript (javascript)

Put it all together.

function compareBy(propertyName) {

return function (a, b) {

let x = a[propertyName],

y = b[propertyName];

if (x > y) {

return 1;

} else if (x < y) {

return -1;

} else {

return 0;

}

};

}

let products = [

{ name: 'iPhone', price: 900 },

{ name: 'Samsung Galaxy', price: 850 },

{ name: 'Sony Xperia', price: 700 },

];

*// sort products by name*

console.log('Products sorted by name:');

products.sort(compareBy('name'));

console.table(products);

*// sort products by price*

console.log('Products sorted by price:');

products.sort(compareBy('price'));

console.table(products);

Code language: JavaScript (javascript)

More JavaScript Functions are First-Class Citizens example

The following example defines two functions that convert a length in centimeters to inches and vice versa:

function cmToIn(length) {

return length / 2.54;

}

function inToCm(length) {

return length \* 2.54;

}

Code language: JavaScript (javascript)

The following convert() function has two parameters. The first parameter is a function and the second one is the length that will be converted based on the first argument:

function convert(fn, length) {

return fn(length);

}

Code language: JavaScript (javascript)

To convert cm to in, you can call the convert() function and pass the cmToIn function into the convert() function as the first argument:

let inches = convert(cmToIn, 10);

console.log(inches);

Code language: JavaScript (javascript)

Output:

3.937007874015748

Code language: CSS (css)

Similarly, to convert a length from inches to centimeters, you can pass the inToCm function into the convert() function, like this:

let cm = convert(inToCm, 10);

console.log(cm);

Code language: JavaScript (javascript)

Output:

25.4

Code language: CSS (css)

Put it all together.

function cmToIn(length) {

return length / 2.54;

}

function inToCm(length) {

return length \* 2.54;

}

function convert(fn, length) {

return fn(length);

}

let inches = convert(cmToIn, 10);

console.log(inches);

let cm = convert(inToCm, 10);

console.log(cm);

Code language: JavaScript (javascript)

Output:

3.937007874015748

25.4

Code language: CSS (css)

Summary

* Functions are first-class citizens in JavaScript.
* You can pass functions to other functions as arguments, return them from other functions as values, and store them in variables.

JavaScript Anonymous Functions

**Summary**: in this tutorial, you will learn about JavaScript anonymous functions.

Introduction to JavaScript anonymous functions

An anonymous function is a [function](https://www.javascripttutorial.net/javascript-function/) without a name. The following shows how to define an anonymous function:

(function () {

*//...*

});

Code language: JavaScript (javascript)

Note that if you don’t place the anonymous function inside the (), you’ll get a syntax error. The () makes the anonymous function an expression that returns a function object.

An anonymous function is not accessible after its initial creation. Therefore, you often need to assign it to a variable.

For example, the following shows an anonymous function that displays a message:

let show = function() {

console.log('Anonymous function');

};

show();

Code language: JavaScript (javascript)

In this example, the anonymous function has no name between the function keyword and parentheses ().

Because we need to call the anonymous function later, we assign the anonymous function to the show variable.

Since the whole assignment of the anonymous function to the show variable makes a valid expression, you don’t need to wrap the anonymous function inside the parentheses ().

Using anonymous functions as arguments

In practice, you often pass anonymous functions as arguments to other functions. For example:

setTimeout(function() {

console.log('Execute later after 1 second')

}, 1000);

Code language: JavaScript (javascript)

In this example, we pass an anonymous function into the [setTimeout()](https://www.javascripttutorial.net/javascript-bom/javascript-settimeout/) function. The setTimeout() function executes this anonymous function one second later.

Note that [functions are](https://www.javascripttutorial.net/javascript-functions-are-first-class-citizens/)first-class citizens in JavaScript. Therefore, you can pass a function to another function as an argument.

Immediately invoked function execution

If you want to create a function and execute it immediately after the declaration, you can declare an anonymous function like this:

(function() {

console.log('IIFE');

})();

Code language: JavaScript (javascript)

How it works.

First, define a function expression:

(function () {

console.log('Immediately invoked function execution');

})

Code language: JavaScript (javascript)

This expression returns a function.

Second, call the function by adding the trailing parentheses ():

(function () {

console.log('Immediately invoked function execution');

})();

Code language: JavaScript (javascript)

and sometimes, you may want to pass arguments into it, like this:

let person = {

firstName: 'John',

lastName: 'Doe'

};

(function () {

console.log(person.firstName} + ' ' + person.lastName);

})(person);

Code language: JavaScript (javascript)

Arrow functions

ES6 introduced [arrow function](https://www.javascripttutorial.net/es6/javascript-arrow-function/) expressions that provide a shorthand for declaring anonymous functions:

For example, this function:

let show = function () {

console.log('Anonymous function');

};

Code language: JavaScript (javascript)

… can be shortened using the following arrow function:

let show = () => console.log('Anonymous function');

Code language: JavaScript (javascript)

Similarly, the following anonymous function:

let add = function (a, b) {

return a + b;

};

Code language: JavaScript (javascript)

… is functionally equivalent to the following arrow function:

let add = (a, b) => a + b;

Code language: JavaScript (javascript)

Summary

* Anonymous functions are functions without names.
* Anonymous functions can be used as an argument to other functions or as an immediately invoked function execution.

Understanding JavaScript Pass-By-Value

**Summary**: this tutorial explains how JavaScript pass-by-value works and gives you some examples of passing primitive and reference values to a function.

Before going forward with this tutorial, you should have good knowledge of the[primitive and reference values, and the differences between them](https://www.javascripttutorial.net/javascript-primitive-vs-reference-values/).

JavaScript pass-by-value or pass-by-reference

In JavaScript, all [function](https://www.javascripttutorial.net/javascript-function/) arguments are always passed by value. It means that JavaScript copies the values of the [variables](https://www.javascripttutorial.net/javascript-variables/) into the function arguments.

Any changes that you make to the arguments inside the function do not reflect the passing variables outside of the function. In other words, the changes made to the arguments are not reflected outside of the function.

If function arguments are passed by reference, the changes of variables that you pass into the function will be reflected outside the function. This is not possible in JavaScript.

Pass-by-value of primitives values

Let’s take a look at the following example.

function square(x) {

x = x \* x;

return x;

}

let y = 10;

let result = square(y);

console.log(result); *// 100*

console.log(y); *// 10 -- no change*

Code language: JavaScript (javascript)

How the script works.

First, define a square() function that accepts an argument x. The function assigns the square of x to the x argument.

Next, declare the variable y and initialize its value to 10:

Then, pass the y variable into the square() function. When passing the variable y to the square() function, JavaScript copies y value to the x variable.

After that, the square() function changes the x variable. However, it does not impact the value of the y variable because x and y are separate variables.

Finally, the value of the y variable does not change after the square() function completes.

If JavaScript used the pass-by-reference, the variable y would change to 100 after calling the function.

Pass-by-value of reference values

It’s not obvious to see that reference values are also passed by values. For example:

let person = {

name: 'John',

age: 25,

};

function increaseAge(obj) {

obj.age += 1;

}

increaseAge(person);

console.log(person);

Code language: JavaScript (javascript)

How the script works:

First, define the person variable that references an object with two properties name and age:

Next, define the increaseAge() function that accepts an object obj and increases the age property of the obj argument by one.

Then, pass the person object to the increaseAge() function:

Internally, the JavaScript engine creates the obj reference and make this variable reference the same object that the person variable references.

After that, increase the age property by one inside the increaseAge() function via the obj variable

Finally, accessing the object via the person reference:

It seems that JavaScript passes an object by reference because the change to the object is reflected outside the function. However, this is not the case.

In fact, when passing an object to a function, you are passing the reference of that object, not the actual object. Therefore, the function can modify the properties of the object via its reference.

However, you cannot change the reference passed into the function. For example:

let person = {

name: 'John',

age: 25,

};

function increaseAge(obj) {

obj.age += 1;

*// reference another object*

obj = { name: 'Jane', age: 22 };

}

increaseAge(person);

console.log(person);

Code language: JavaScript (javascript)

Output:

{ name: 'John', age: 26 }

Code language: CSS (css)

In this example, the increaseAage() function changes the age property via the obj argument:

and makes the obj reference another object:

However, the person reference still refers to the original object whose the age property changes to 26. In other words, the increaseAge() function doesn’t change the person reference.

If this concept still confuses you, you can consider function arguments as local variables.

Summary

* JavaScript passes all arguments to a function by values.
* Function arguments are local variables in JavaScript.

JavaScript Recursive Function

**Summary**: in this tutorial, you will learn how to use the recursion technique to develop a JavaScript recursive function, which is a function that calls itself.

Introduction to the JavaScript recursive functions

A recursive function is a [function](https://www.javascripttutorial.net/javascript-function/) that calls itself until it doesn’t. And this technique is called recursion.

Suppose that you have a function called recurse(). The recurse() is a recursive function if it calls itself inside its body, like this:

function recurse() {

*// ...*

recurse();

*// ...*

}

Code language: JavaScript (javascript)

A recursive function always has a condition to stop calling itself. Otherwise, it will call itself indefinitely. So a recursive function typically looks like the following:

function recurse() {

if(condition) {

*// stop calling itself*

*//...*

} else {

recurse();

}

}

Code language: JavaScript (javascript)

Generally, you use recursive functions to break down a big problem into smaller ones. Typically, you will find the recursive functions in data structures like binary trees and graphs and algorithms such as binary search and quicksort.

JavaScript recursive function examples

Let’s take some examples of using recursive functions.

1) A simple JavaScript recursive function example

Suppose that you need to develop a function that counts down from a specified number to 1. For example, to count down from 3 to 1:

3

2

1

The following shows the countDown() function:

function countDown(fromNumber) {

console.log(fromNumber);

}

countDown(3);

Code language: JavaScript (javascript)

This countDown(3) shows only the number 3.

To count down from the number 3 to 1, you can:

1. show the number 3.
2. and call the countDown(2) that shows the number 2.
3. and call the countDown(1) that shows the number 1.

The following changes the countDown() to a recursive function:

function countDown(fromNumber) {

console.log(fromNumber);

countDown(fromNumber-1);

}

countDown(3);

Code language: JavaScript (javascript)

This countDown(3) will run until the call stack size is exceeded, like this:

Uncaught RangeError: Maximum call stack size exceeded.

Code language: JavaScript (javascript)

… because it doesn’t have the condition to stop calling itself.

The count down will stop when the next number is zero. Therefore, you add an [if condition](https://www.javascripttutorial.net/javascript-if/) as follows:

function countDown(fromNumber) {

console.log(fromNumber);

let nextNumber = fromNumber - 1;

if (nextNumber > 0) {

countDown(nextNumber);

}

}

countDown(3);

Code language: JavaScript (javascript)

Output:

3

2

1

The countDown() seems to work as expected.

However, as mentioned in the [Function type tutorial](https://www.javascripttutorial.net/javascript-function-type/), the function’s name is a reference to the actual function object.

If the function name is set to null somewhere in the code, the recursive function will stop working.

For example, the following code will result in an error:

let newYearCountDown = countDown;

*// somewhere in the code*

countDown = null;

*// the following function call will cause an error*

newYearCountDown(10);

Code language: JavaScript (javascript)

Error:

Uncaught TypeError: countDown is not a function

Code language: JavaScript (javascript)

How the script works:

* First, assign the countDown function name to the variable newYearCountDown.
* Second, set the countDown function reference to null.
* Third, call the newYearCountDown function.

The code causes an error because the body of the countDown() function references the countDown function name, which was set to null at the time of calling the function.

To fix it, you can use a named function expression as follows:

let countDown = function f(fromNumber) {

console.log(fromNumber);

let nextNumber = fromNumber - 1;

if (nextNumber > 0) {

f(nextNumber);

}

}

let newYearCountDown = countDown;

countDown = null;

newYearCountDown(10);

Code language: JavaScript (javascript)

2) Calculate the sum of n natural numbers example

Suppose you need to calculate the sum of natural numbers from 1 to n using the recursion technique. To do that, you need to define the sum() recursively as follows:

sum(n) = n + sum(n-1)

sum(n-1) = n - 1 + sum(n-2)

...

sum(1) = 1

The following illustrates the sum() recursive function:

function sum(n) {

if (n <= 1) {

return n;

}

return n + sum(n - 1);

}

Code language: JavaScript (javascript)

Summary

* A recursive function is a function that calls itself until it doesn’t
* A recursive function always has a condition that stops the function from calling itself.

JavaScript Default Parameters

**Summary**: in this tutorial, you will learn how to handle JavaScript default parameters in ES6.

TL;DR

function say(message='Hi') {

console.log(message);

}

say(); *// 'Hi'*

say('Hello') *// 'Hello'*

Code language: JavaScript (javascript)

The default value of the message paramater in the say() function is 'Hi'.

In JavaScript, default function parameters allow you to initialize named parameters with default values if no values or undefined are passed into the function.

Arguments vs. Parameters

Sometimes, you can use the terms argument and parameter interchangeably. However, by definition, parameters are what you specify in the [function declaration](https://www.javascripttutorial.net/javascript-function/) whereas the arguments are what you pass into the function.

Consider the following add() function:

function add(x, y) {

return x + y;

}

add(100,200);

Code language: JavaScript (javascript)

In this example, the x and y are the parameters of the add() function, and the values passed to the add() function 100 and 200 are the arguments.

Setting JavaScript default parameters for a function

In JavaScript, a parameter has a default value of [undefined](https://www.javascripttutorial.net/javascript-data-types/#undefined). It means that if you don’t pass the arguments into the [function](https://www.javascripttutorial.net/javascript-function/), its parameters will have the default values of undefined.

See the following example:

function say(message) {

console.log(message);

}

say(); *// undefined*

Code language: JavaScript (javascript)

The say() function takes the message parameter. Because we didn’t pass any argument into the say() function, the value of the message parameter is undefined.

Suppose that you want to give the message parameter a default value 10.

A typical way for achieving this is to test parameter value and assign a default value if it is undefined using a [ternary operator](https://www.javascripttutorial.net/javascript-ternary-operator/):

function say(message) {

message = typeof message !== 'undefined' ? message : 'Hi';

console.log(message);

}

say(); *// 'Hi'*

Code language: JavaScript (javascript)

In this example, we didn’t pass any value into the say() function. Therefore, the default value of the message argument is undefined. Inside the function, we reassigned the message variable the Hi string.

ES6 provides you with an easier way to set the default values for the function parameters like this:

function fn(param1=default1, param2=default2,..) {

}

Code language: JavaScript (javascript)

In the syntax above, you use the [assignment operator](https://www.javascripttutorial.net/javascript-assignment-operators/) (=) and the default value after the parameter name to set a default value for that parameter. For example:

function say(message='Hi') {

console.log(message);

}

say(); *// 'Hi'*

say(undefined); *// 'Hi'*

say('Hello'); *// 'Hello'*

Code language: JavaScript (javascript)

How it works.

* In the first function call, we didn’t pass any argument into the say() function, therefore message parameter took the default value 'Hi'.
* In the second function call, we passed the undefined into the say() function, hence the message parameter also took the default value 'Hi'.
* In the third function call, we passed the 'Hello' string into the say() function, therefore message parameter took the string 'Hello' as the default value.

More JavaScript default parameter examples

Let’s look at some more examples to learn some available options for setting default values of the function parameters.

1) Passing undefined arguments

The following createDiv() function creates a new <div> element in the document with a specific height, width, and border-style:

function createDiv(height = '100px', width = '100px', border = 'solid 1px red') {

let div = document.createElement('div');

div.style.height = height;

div.style.width = width;

div.style.border = border;

document.body.appendChild(div);

return div;

}

Code language: JavaScript (javascript)

The following doesn’t pass any arguments to the function so the createDiv() function uses the default values for the parameters.

createDiv();

Suppose you want to use the default values for the height and width parameters and specific border style. In this case, you need to pass undefined values to the first two parameters as follows:

createDiv(undefined,undefined,'solid 5px blue');

Code language: JavaScript (javascript)

2) Evaluating default parameters

JavaScript engine evaluates the default arguments at the time you call the function. See the following example:

function put(toy, toyBox = []) {

toyBox.push(toy);

return toyBox;

}

console.log(put('Toy Car'));

*// -> ['Toy Car']*

console.log(put('Teddy Bear'));

*// -> ['Teddy Bear'], not ['Toy Car','Teddy Bear']*

Code language: JavaScript (javascript)

The parameter can take a default value which is a result of a function.

Consider the following example:

function date(d = today()) {

console.log(d);

}

function today() {

return (new Date()).toLocaleDateString("en-US");

}

date();

Code language: JavaScript (javascript)

The date() function takes one parameter whose default value is the returned value of the today() function. The today() function returns today’s date in a specified string format.

When we declared the date() function, the today() function has not yet evaluated until we called the date() function.

We can use this feature to make arguments mandatory. If the caller doesn’t pass any argument, we throw an error as follows:

function requiredArg() {

throw new Error('The argument is required');

}

function add(x = requiredArg(), y = requiredArg()){

return x + y;

}

add(10); *// error*

add(10,20); *// OK*

Code language: JavaScript (javascript)

3) Using other parameters in default values

You can assign a parameter a default value that references other default parameters as shown in the following example:

function add(x = 1, y = x, z = x + y) {

return x + y + z;

}

console.log(add()); *// 4*

Code language: JavaScript (javascript)

In the add() function:

* The default value of the y is set to x parameter.
* The default value of the z is the sum of x and y
* The add() function returns the sum of x, y, and z.

The parameter list seems to have its own [scope](https://www.javascripttutorial.net/javascript-variable-scope/). If you reference the parameter that has not been initialized yet, you will get an error. For example:

function subtract( x = y, y = 1 ) {

return x - y;

}

subtract(10);

Code language: JavaScript (javascript)

Error message:

Uncaught ReferenceError: Cannot access 'y' before initialization

Code language: JavaScript (javascript)

Using functions

You can use a return value of a function as a default value for a parameter. For example:

let taxRate = () => 0.1;

let getPrice = function( price, tax = price \* taxRate() ) {

return price + tax;

}

let fullPrice = getPrice(100);

console.log(fullPrice); *// 110*

Code language: JavaScript (javascript)

In the getPrice() function, we called the taxRate() function to get the tax rate and use this tax rate to calculate the tax amount from the price.

The arguments object

The value of the arguments object inside the function is the number of actual arguments that you pass to the function. For example:

function add(x, y = 1, z = 2) {

console.log( arguments.length );

return x + y + z;

}

add(10); *// 1*

add(10, 20); *// 2*

add(10, 20, 30); *// 3*

Code language: JavaScript (javascript)

Now, you should understand the JavaScript default function parameters and how to use them effectively.