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'
```

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 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);
```

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. It means that if you don't pass the arguments into the function, its parameters will have the default values of `undefined` .

See the following example:

```
function say(message) {
    console.log(message);
}

say(); // undefined
```

The `say()` function takes the `message` parameter. Since we don'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:

```
function say(message) {
    message = typeof message !== 'undefined' ? message : 'Hi';
    console.log(message);
}
say(); // 'Hi'
```

In this example, we don'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,..) {
}
```

In the syntax above, you use the assignment operator (`=`) 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'
```

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;
}
```

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');
```

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']
```

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();
```

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
```

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
```

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. 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);
```

Error message:

```
Uncaught ReferenceError: Cannot access 'y' before initialization
```

Using functions

You can use the 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
```

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
```

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

Quiz