5 Ways to Write Functions in JavaScript

dev.to/koladev/5-ways-to-write-functions-in-javascript-17d5

Mangabo Kolawole

Mangabo Kolawole
Posted on Aug 1, 2021 • Updated on Aug 9, 2022

#javascript #frontend #beginners

A function is a block of organized reusable code used to perform a single action.

Like many programming languages such as JavaScript, you can add and reusable code used to perform a single action using many ways.

This post will teach you seven approaches to write JavaScript functions: syntax and some examples.
I'll also discuss when you can use the different types efficiently, and also the pros and the cons.

Table of content

- 5 - Arrow function

1 - Function Declaration

A function declaration is the most common way to define a function in JavaScript.

> To declare a function, you use the `function` keyword followed by an obligatory function name, a list of parameters in brackets, and a pair of curly braces to write the function code.

```
function nameOfTheFunction(param1, param2, ...){
    console.log("Something")
   // line1
   ...
}
```

Example

```
function isPositive(number){
    return number > 0;
}
```

The function `isPositive()` defines a variable `isPositive` in the current scope of execution with the identifier equal to the function name. That means that the variable `isPositive` holds the function object.

Function hoisting

One of the most important properties of the function declaration is the hoisting mechanism. It allows using the function before the declaration in the same scope.

Example

```
multiplyNumbers(5,5)

function multiplyNumbers(a, b){
    return a * b;
}
```

Note that to execute a declared function, you'll need to invoke it. As you saw, you just need to write the name of the function followed by brackets containing the arguments if necessary.

2 - Function Expression

A function expression is closely similar to the syntax of the function declaration.

> Using `var`, `const`, or `let` keyword, followed by the function name and affectation sign (`=` in JavaScript), you enter the keyword function followed by parameters in brackets and a pair of curly braces containing the function code.

```
const functionName = function(param1, param2, ...){
    //code
}
```

Here is a clear example :

```
const sum = function(a, b){
    return a + b;
}
sum(5, 6); // => 11
```

Function expressions are very useful if you want to write methods within objects.

```
const methods = {
    sum: function(a, b){
        return a + b;
    },
    substract: function(a, b){
        return a - b;
    }
}

methods.sum(5,6); // => 11
methods.substract(5,6); // => -1
```

Contrary to function declaration allowing **hoisting**, you can't call your function if you haven't defined it yet.

One of the main advantages of using a function expression is easy debugging. When your program will encounter an error, the stack trace will contain the name of the function.

3 - Shorthand function

Shorthand syntax has been introduced with ES2015 and is quite similar to the getter, setter syntax.

```
const obj = {
    items:[],
    get(index){
        return this.items[index];
    },
    set(...elements){
        this.items.push(...elements)
    }
}

items.add("foo", "bar");
items.get(1) // => "bar"
```

> You can define them using the function name, followed by a list of parameters in a pair of parenthesis `()` followed by a pair of curly braces that delimits the body statements.

This function syntax is very common when working with objects. You call the function like this :

`object.functionName(...parameters)`.

Pros

- Shorter syntax is easier to read;
- Name functions are created, contrary to function expression;

4 - Constructors

In JavaScript, a constructor is a function used to create objects.

> To create a constructor, you use the keyword function followed by the name of your function, always **camelCased** by convention, followed by a pair of parenthesis where you can add parameters, followed by a pair of curly braces describing the code logic.

Example

```
function shoes(size, mark){
    this.size = size;
    this.mark = mark;
};

let yeezy = new shoes(37, adidas);
console.log(yeezy.size); => 37
console.log(yeezy.mark); => 'adidas'
```

One of the most important things to notice here is the usage of the `this` and `new`. Here `this` refers to the `object` when this `object` is created.

```
function vegetal(){
    this.type = "vegetal";
};

let tomato = new vegetal();

console.log(tomato.type); => "vegetal"
```

To create an object from a constructor function, we use the `new` keyword.

Pros

Can be used to create multiples objects that can be mutated without changing the parent. In this case, it can be effective than object expression.

Example

```
let vegetal = {
    this.type = "vegetal";
};

let tomato = vegetal();
tomato.type = "Fruit";
console.log(tomato.type); //=> "Fruit"
```

If you want to add a property to objects derived from your constructor, you can do it easily.

Example

```
function vegetal(){
    this.type = "vegetal";
}

let tomato = new vegetal();
let orange = new vegetal();
tomato.type = "Fruit";
orange.juice = true;

console.log(tomato.type); //=> "Fruit"
console.log(orange.juice); //=> true
console.log(vegetal.type, vegetal.juice); // => undefined, undefined
```

If you want to add a new property to the constructor, just use `Object.prototype.property`.

Example

```
function vegetal(){
    this.type = "vegetal";
}

let tomato = new vegetal();
let orange = new vegetal();
console.log(tomato.type, orange.type); //=> vegetal, vegetal

vegetal.prototype.isHealthy = true;
console.log(tomato.isHealthy, orange.isHealthy); //=> true, true
```

5 - Arrow Functions

Array functions is one of the most used features introduced in ES6. It allows developers to create functions in a cleaner way contrary to the function declaration.

```
let sum = (a,b) => a+b;

let sum = function (a,b){
    return a + b;
};

function person(){
  this.name = "John";
  showName = () => {
        console.log(this.name);
    }
}
let someone = new person()
```

Now, it's important to inform you: `this` keyword is a little complex with arrow functions. With regular function, `this` keyword represents the object that is called the function.

It can be the window, the document, or a button for example.

However, with the arrow function, the `this` keyword always represents the object that defined the arrow function.

Okay, it sounds a little bit complex. Let's see with some examples we'll try in the browser console :

```
// Declaring a normal function
function greetings() {
    console.log("Hello " + this);
};
greetings(); // => "Hello [object Window]"
```

Now, let's use an arrow function.

```
const greetings = () => console.log("Hello " + this);

greetings(); // => "Hello [object Window]"
```

The same result right ... And that's normal. `this` here represents the Object Window, because the function is defined in Object Window object scope.
Let's create an arrow function with our own object.

```
const greetings = {
    greetUser: null,
    hello: function () {
        this.greetUser = () => { console.log(this) };
    }
};

greetings.hello(); // To initialize greetings.greetUser with an arrow function
greetings.greetUser(); // => Object { greetUser: greetUser(), hello: hello() }
```

Pros

- Reduces a lot of code and makes it more readable. Actually, the arrow functions are very effective when used for callbacks.
- Having contextual `this`

Cons

> Avoid using arrow functions for event handlers, object methods prototype methods, or functions that use the `arguments` object

Conclusion

In this article, we learned about 5 ways to write functions in JavaScript. There are more ways to write functions in JavaScript but for the sake of simplicity, I decided to stick to the most common functions you'll encounter on your journey.
But here are some resources if you want to know more:

> new Function

And as every article can be made better so your suggestion or questions are welcome in the comment section. 😉

Top comments (8)

▼ ⤫

- Aug 4 '21

There is also Function constructor:

```
var sum = new Function('a,b', 'return a + b');
```

Note that Functions are instances of Function.

```
(function() {}) instanceof Function;
```

- Aug 2 '21

"Avoid using arrow functions for event handlers"
Why that? I do that quite often, that's one big advantage of Es6 to use arrow functions for event handlers

- Aug 3 '21

He said "that use the `arguments`" object. It's not defined for arrow functions. Also, they don't have their own `this` but keep the one from their parent function, something that in certain cases is very useful.

Trending in JavaScript

The JavaScript community is focusing on **React** mastery, interview preparation, and **neural networks** in vanilla JS. There's also interest in contributing to open-source repositories and frontend best practices.

44 React Frontend Interview Questions

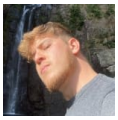
Yan Levin ・ Oct 12

#webdev #javascript #beginners #react

🧠 An AI / neural network...in vanilla JS! 😱 With no libraries! 🤯

GrahamTheDev ・ Oct 13

#ai #javascript #beginners #webdev



Don’t do it on Frontend or... Frontend good practices for devs

Lucas Menezes ・ Oct 19

#webdev #frontend #javascript #beginners