Ashish Oli · Follow

Jul 19, 2021 · 4 min read

Function declaration, expression, and arrow functions in Javascript.

Sometimes people confuse function declaration, function expression, and arrow function, but trust me it is not confusing as you think. Let us break it into fundamentals and understand them.

Let us first start with the syntax:-

NOTE:- using es6 ‘let’ and ‘const’ and not ‘var’.

Function Declaration.

```
/*
    - Function Declaration
    - functions created using 'function' keyword,
    - are called function declaration 👇.

*/

function test() {
    console.log("This is function declaration");
}
test();
```

Function Expression

```
*
    - Function Expression
    - function when assigned to a variable is called function expression,
    - like below 👇
*/

let test = function () {
    console.log("this is function expression");
};

test();
```

Arrow Function.

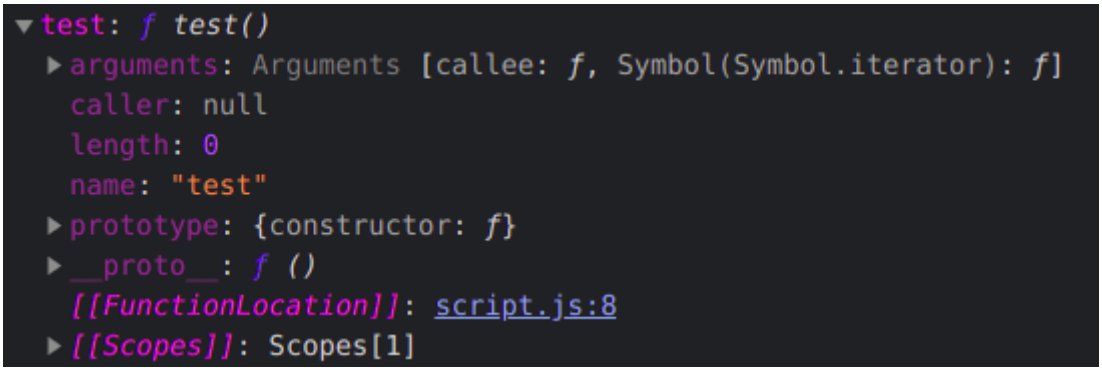
```
/*
    - Arrow Functions
    - this functions are similar to function expression with slight different syntax,
    - but with much more functianlity inside
    - 👇
*/

let test = () => {
    console.log("this is syntax for arrow function");
};

test();
```

Now as you know the syntax let us understand the functionality of each.

Hoisting

There is a memory component inside the execution context in javascript. The role of this memory component is to allocate the memory to each variable and function in the script before actually running the code, this is a kind of preflight mechanism that takes place and assigns memory to all the variables and functions in the script before its actual execution.

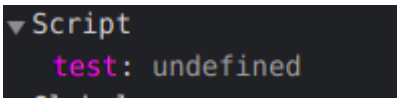
Function declaration —



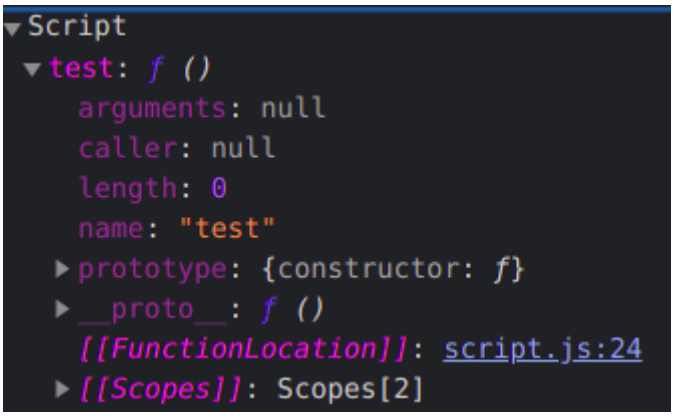
As you can see that in the global object the **test** function is created and contains the whole body of the function.

Function expression and arrow function.

The hoisting is the same in the case of function expression and arrow functions.



As it can be seen that the test is assigned a value undefined (it is treated as a normal variable) and the function being a first-class citizen in javascript, it can be assigned to any variable.



Hence we can see that when the code executes the placeholder undefined is replaced with the whole function body.

This is the same with arrow functions as you can compare the above pictures.

But if you notice there is a slight difference between the two of them and that is arrow function is missing the ***prototype*** property.

Hence it brings us to our next point i.e ***this*** keyword.

this is an object in javascript which maps to the scope of the execution context of the script.

Now let us find out the pros and cons between all of the function cases and know where to use which one.

- first of all, as you know saw arrow function is missing prototype property hence it cannot be used to create a constructor.

```
var Foo = () => {};
var foo = new Foo(); // TypeError: Foo is not a constructor
```

- An arrow function does not have its own bindings to `this` or `super`, and should not be used as `methods`.
- An arrow function does not have, `arguments`, or `new.target` keywords.
- Arrow functions are not suitable for `call`, `apply` and `bind` methods, which generally rely on establishing a scope.
- In regular function declaration and expression, we require a ***return*** keyword to return something like —

```
function test() {
    "any value or expression";
}
test(); // undefined - because no return statement
```

```
const test = () => "any value or expression";

test(); // any value or expression
```

- but in arrow functions, we don't need any ***return*** if we return a single line value or expression.
- arrow function provides clean code and readability.

Hence all the function declaration types have their own pros and cons and are used widely for their respective use cases, use them according to your use cases.

Next time when you are declaring functions with any of the above types never miss these points to write better code.

useful links: this, arrow function, functions

Thank you for reading.