**JavaScript Tutorial: Arrow Functions In JavaScript | Web Development Tutorials #60**

In this *tutorial*, we are going to learn about *Arrow Functions* and *Lexical this* used in JavaScript. Make a new file as *tut60.html* and add an instant boilerplate to get the basic HTML code. Then give the title as **Arrow Functions** under the <title> tag.

An **arrow function expression** is a syntactically compact alternative to a *regular function expression*, although without its own bindings to *this, arguments, super or new.target*keywords. Arrow function expressions are ill-suited as methods, and they cannot be used as constructors.

Let us understand by example. If we simply write the below JavaScript-

function greet() {

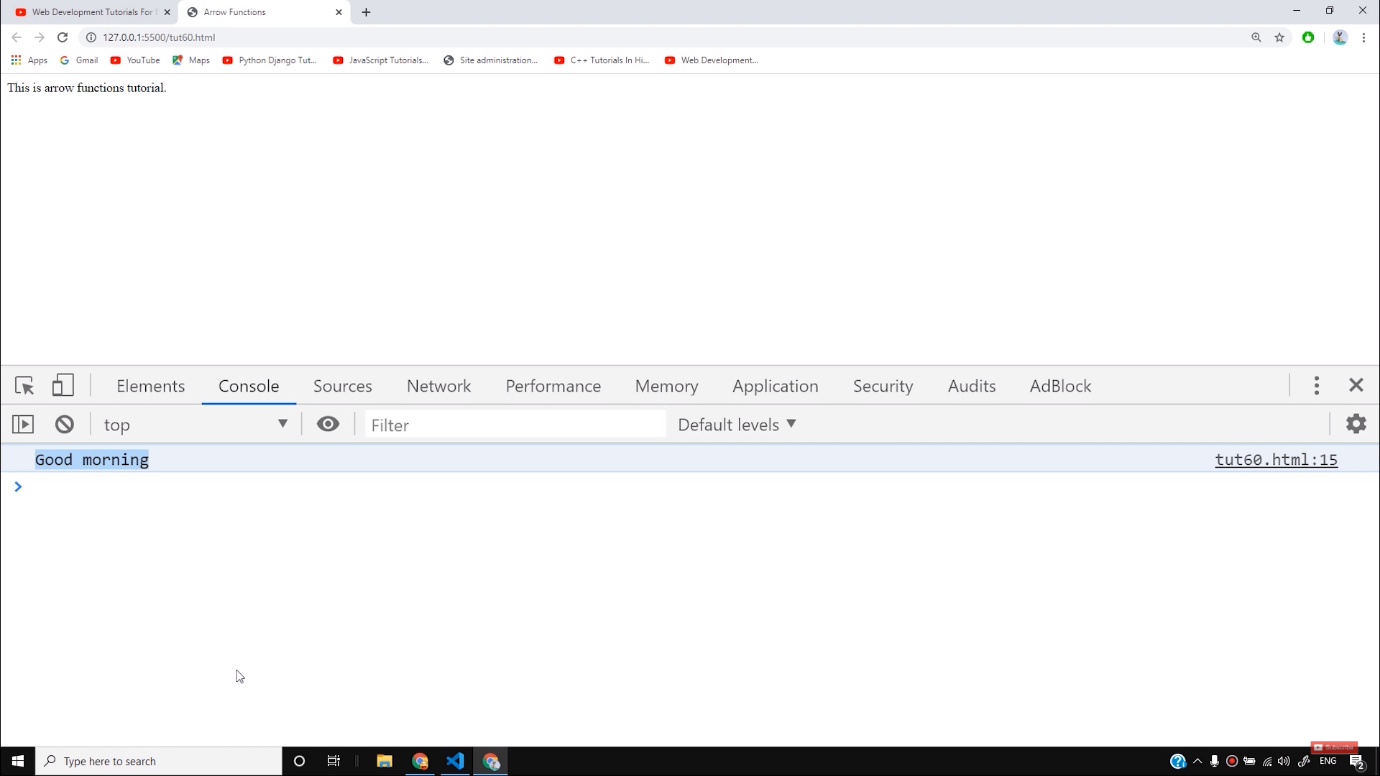
console.log('Good morning');

}

greet();

Copy

The output we get is as follows-



But if we write the same function with the help of *arrow functions* then our work will reduce as follows-

// Arrow function

let greet = ()=> {

console.log('Good morning');

}

Copy

We will get the same output as the above after executing this code. This function can also be used with the **setTimeout()**function as follows-

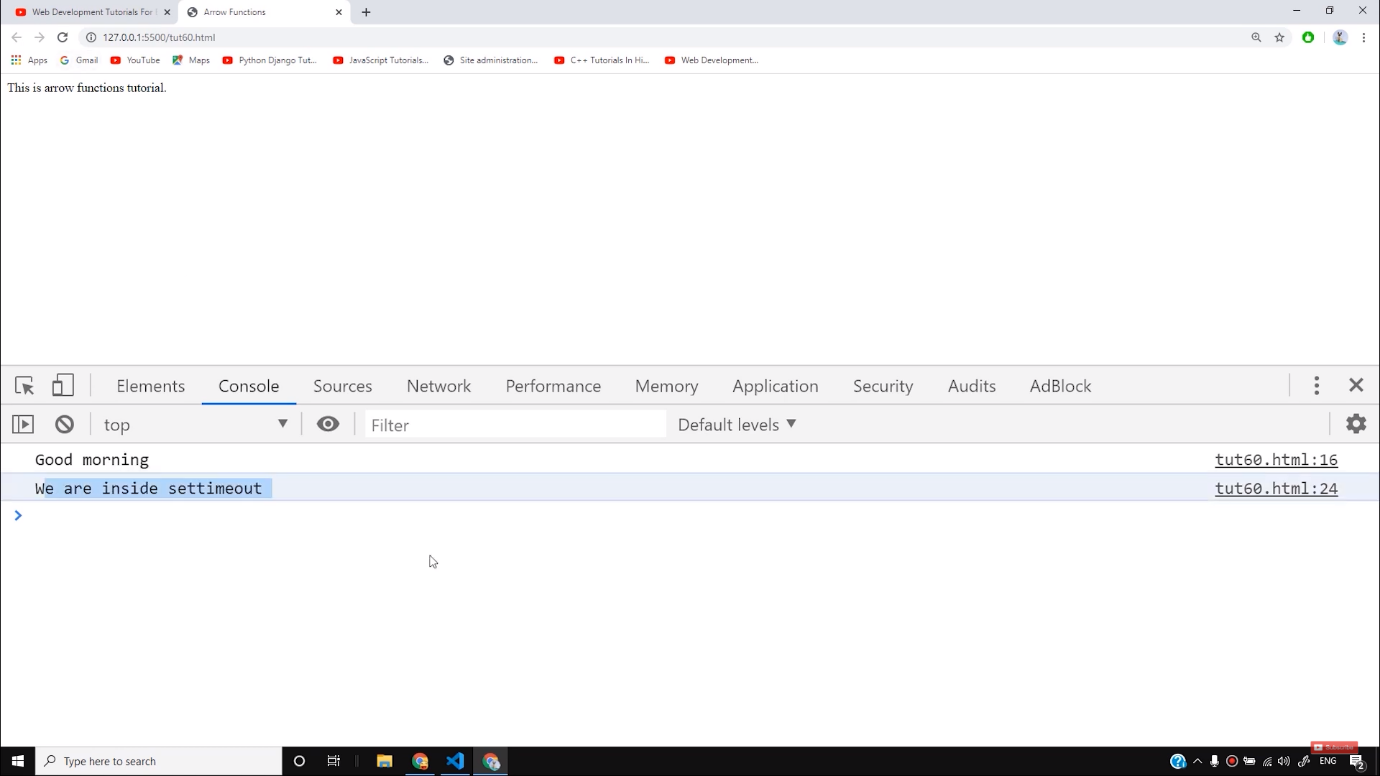
setTimeout(() => {

console.log("We are inside settimeout");

}, 3000);

Copy

With the help of arrow functions, it is not necessary to define the whole function and its components. We will get the output after 3 seconds as follows-



We can also simply add the two numbers with the arrow functions as follows-

let sum2 = (a, b) => a+b;

Copy

If we have any single variable in our code, then we don’t need to put any braces also. The example for this is as follows-

let half = a => a/2;

Copy

This function will return half the value of *a*.

Now let us see the use of *lexical this* with the help of an example. If we make the object *obj1* and an array of *names* and then make a function as *speak*which does the following task as follows-

let obj1={

names: ["Harry", "Rohan", "SkillF", "DjKhiladi"],

speak(){

this.names.forEach((student)=>{

console.log("Kukdoo Koo " + student);

});

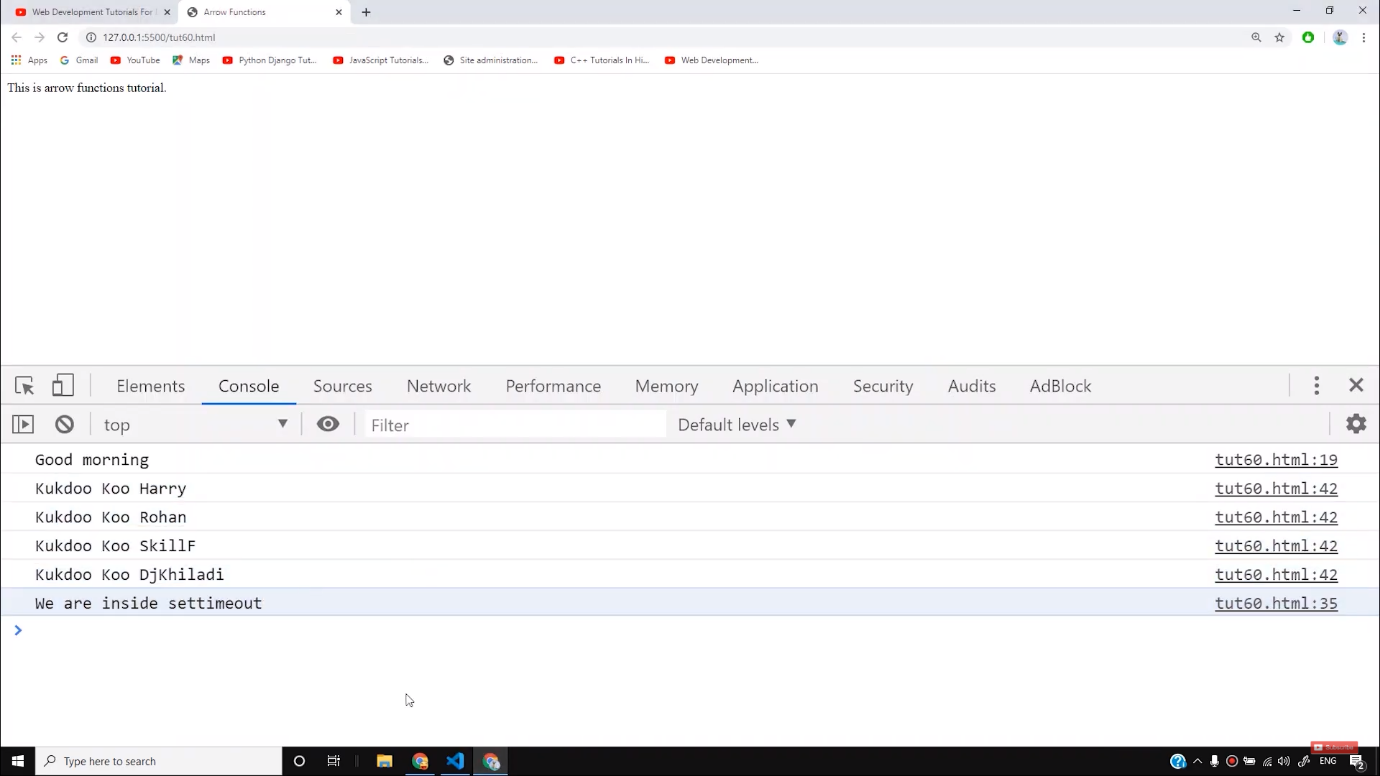
}

}

obj1.speak();

Copy

We know that after running this function, we will get the expected output as follows-



In the above example, we have made a *speak* function, add a *forEach()*loop to it, and then added an *arrow function.* But if we add a variable greeting and add *this.greeting* inside the console.log as follows-

let obj1={

greeting: "Good Morning",

names: ["Harry", "Rohan", "SkillF", "DjKhiladi"],

speak(){

this.names.forEach((student)=>{

console.log(this.greeting + " Kukdoo Koo " + student);

});

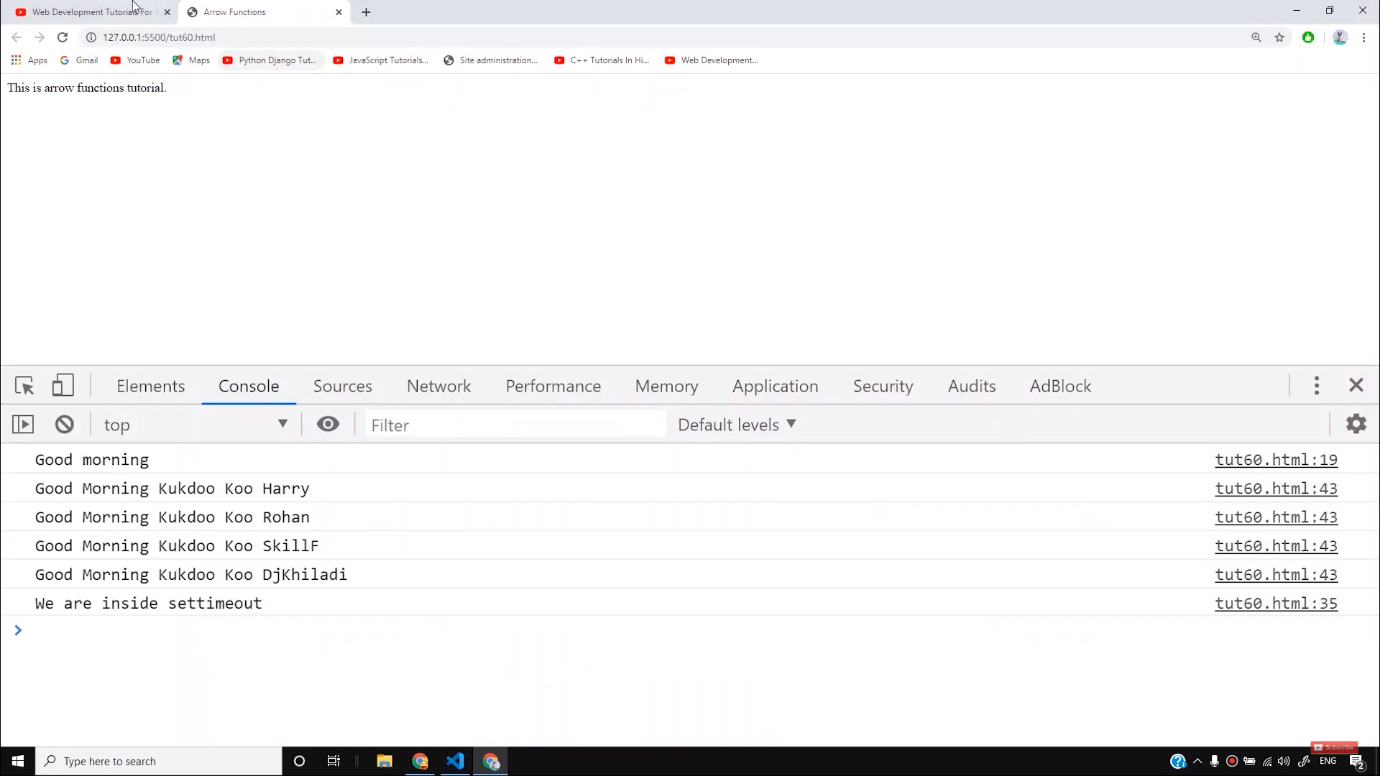
}

}

obj1.speak();

Copy

*This* variable is normally used to point the variable inside the function. But in the case of arrow functions, it points the variable outside the function. This is known as *lexical this.*Therefore, we get the output as follows-



So I believe, you must have understood the concept of *arrow functions* and *lexical this* with the help of this tutorial.

**Code as described/written in the video**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Arrow Functions</title>

</head>

<body>

<div class="container">

This is arrow functions tutorial.

</div>

<script>

// Arrow function

// let greet = ()=> {

// console.log('Good morning');

// }

let greet = () => console.log('Good morning');

// let sum2 = (a, b)=>{

// return a+b;

// };

let sum2 = (a, b) => a+b;

let half = a => a/2;

// function greet() {

// console.log('Good morning');

// }

greet();

setTimeout(() => {

console.log("We are inside settimeout");

}, 3000);

let obj1={

greeting: "Good Morning",

names: ["Harry", "Rohan", "SkillF", "DjKhiladi"],

speak(){

this.names.forEach((student)=>{

console.log(this.greeting + " Kukdoo Koo " + student);

});

}

}

obj1.speak();

</script>

</body>

</html>

# JavaScript Arrow Function

[❮ Previous](https://www.w3schools.com/js/js_this.asp)[Next ❯](https://www.w3schools.com/js/js_classes.asp)

Arrow functions were introduced in ES6.

Arrow functions allow us to write shorter function syntax:

let myFunction = (a, b) => a \* b;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_arrow_function)

### **Before:**

hello = function() {  
  return "Hello World!";  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_arrow_function1)

### **With Arrow Function:**

hello = () => {  
  return "Hello World!";  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_arrow_function2)

It gets shorter! If the function has only one statement, and the statement returns a value, you can remove the brackets and the return keyword:

### **Arrow Functions Return Value by Default:**

hello = () => "Hello World!";

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_arrow_function3)

**Note:** This works only if the function has only one statement.

If you have parameters, you pass them inside the parentheses:

### **Arrow Function With Parameters:**

hello = (val) => "Hello " + val;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_arrow_function4)

In fact, if you have only one parameter, you can skip the parentheses as well:

### **Arrow Function Without Parentheses:**

hello = val => "Hello " + val;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_arrow_function5" \t "_blank)

ADVERTISEMENT

## **What About this?**

The handling of this is also different in arrow functions compared to regular functions.

In short, with arrow functions there are no binding of this.

In regular functions the this keyword represented the object that called the function, which could be the window, the document, a button or whatever.

With arrow functions the this keyword always represents the object that defined the arrow function.

Let us take a look at two examples to understand the difference.

Both examples call a method twice, first when the page loads, and once again when the user clicks a button.

The first example uses a regular function, and the second example uses an arrow function.

The result shows that the first example returns two different objects (window and button), and the second example returns the window object twice, because the window object is the "owner" of the function.

### **Example**

With a regular function this represents the object that calls the function:

// Regular Function:  
hello = function() {  
  document.getElementById("demo").innerHTML += this;  
}  
  
// The window object calls the function:  
window.addEventListener("load", hello);  
  
// A button object calls the function:  
document.getElementById("btn").addEventListener("click", hello);

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_arrow_function6)

### **Example**

With an arrow function this represents the owner of the function:

// Arrow Function:  
hello = () => {  
  document.getElementById("demo").innerHTML += this;  
}  
  
// The window object calls the function:  
window.addEventListener("load", hello);  
  
// A button object calls the function:  
document.getElementById("btn").addEventListener("click", hello);