# ES6 Arrow Function

Arrow functions are introduced in [ES6](https://www.javatpoint.com/es6), which provides you a more accurate way to write the [functions in JavaScript](https://www.javatpoint.com/javascript-function). They allow us to write smaller function syntax. Arrow functions make your code more readable and structured.

Arrow functions are **anonymous functions** (the functions without a name and not bound with an identifier). They don't return any value and can declare without the function keyword. Arrow functions cannot be used as the constructors. The context within the arrow functions is lexically or statically defined. They are also called as **Lambda Functions** in different languages.

Arrow functions do not include any prototype property, and they cannot be used with the new keyword.

**Syntax for defining the arrow function**

1. **const** functionName = (arg1, arg2, ?..) => {
2. //body of the function
3. }

There are three parts to an Arrow Function or Lambda Function:

* **Parameters:** Any function may optionally have the parameters.
* **Fat arrow notation/lambda notation:** It is the notation for the **arrow (=>).**
* **Statements:** It represents the instruction set of the function.

Let us try to understand it with an example.

In the following example, we are defining three functions that show **Function Expression, Anonymous Function, and Arrow Function.**

1. // function expression
3. var myfun1 = function show() {
4. console.log("It is a Function Expression");
5. }
7. // Anonymous function
9. var myfun2 = function () {
10. console.log("It is an Anonymous Function");
11. }
13. //Arrow function
15. var myfun3 = () => {
16. console.log("It is an Arrow Function");
17. };
19. myfun1();
20. myfun2();
21. myfun3();

**Output**

It is a Function Expression

It is an Anonymous Function

It is an Arrow Function

### Syntactic Variations

There are some syntactic variations for the arrow functions that are as follows:

* **Optional parentheses for the single parameter**

1. var num = x => {
2. console.log(x);
3. }
4. num(140);

**Output**

140

* **Optional braces for single statement and the empty braces if there is not any parameter required.**

1. var show = () => console.log("Hello World");
2. show();

**Output**

Hello World

## Arrow Function with Parameters

If you require to pass more than one parameter by using an arrow function, then you have to pass them within the parentheses.

**For example**

1. var show = (a,b,c) => {
2. console.log(a + " " + b + " " + c );
3. }
4. show(100,200,300);

**Output**

100 200 300

## Arrow function with default parameters

In ES6, the function allows the initialization of parameters with default values, if there is no value passed to it, or it is undefined. You can see the illustration for the same in the following code:

**For example**

var show = (a, b=200) => {

console.log(a + " " + b);

}

show(100);

In the above function, the value of **b** is set to **200** by default. The function will always consider **200** as the value of **b** if no value of **b** is explicitly passed.

**Output**

100 200

The default value of the parameter **'b'** will get overwritten if the function passes its value explicitly. You can see it in the following example:

**For example**

1. var show = (a, b=200) => {
2. console.log(a + " " + b);
3. }
4. show(100,500);

**Output**

100 500

## Arrow function with Rest parameters

Rest parameters do not restrict you to pass the number of values in a function, but all the passed values must be of the same type. We can also say that rest parameters act as the placeholders for multiple arguments of the same type.

For declaring the rest parameter, the parameter name should be prefixed with the spread operator that has three periods (not more than three or not less than three).

You can see the illustration for the same in the following example:

1. var show = (a, ...args) => {
2. console.log(a + " " + args);
3. }
4. show(100,200,300,400,500,600,700,800);

**Output**

100 200,300,400,500,600,700,800

## Arrow Function without Parentheses

If you have a single parameter to pass, then the parentheses are optional.

**For example**

1. var show = x => {
2. console.log(x);
3. }
4. show("Hello World");

**Output**

Hello World

## Advantages of Arrow Functions

There are some advantages of using arrow functions that are shown in the below image:

Let us try to elaborate on the advantages which are available in the above image.

**1. It reduces the code size:** When we use the syntax of arrow function, the size of the code gets reduced. We can write less amount of code by using the arrow function.

**2. Return statement and Functional braces are optional for single line functions:**In ES5, we need to define the return statement in the functions, but in ES6, we do not require to define the return statement for single-line functions. Functional braces are also optional for the single-line functions.

**For example**

**In ES5**

1. function show(value){
2. **return** value/2;
3. }

**In ES6**

1. var show = value => value/2;
2. console.log(show(100));

#### Note: If you are not applying curly braces on the single statement, then you do not require to use return, but if you are using curly braces even on the single statement, then you must have to use the return keyword.

You can understand it in the following example:

**Without Return Keyword**

1. var show = value => {
2. value/2;
3. }
4. console.log(show(50));

**Output**

undefined

**With Return Keyword**

1. var show = value => {
2. **return** value/2;
3. }
4. console.log(show(50));

**Output**

25

**3. Lexically bind the context:** Arrow function binds the context lexically or statically. The handling of **this** is different in arrow functions as compared to regular functions. In the arrow function, there is not any binding of **this.** In regular functions, **this** keyword is used to represent the objects that called the function, which could either be a window, a button, or a document or anything.

But with arrow functions, this keyword always represents the object that defines the arrow function.

Let us try to understand it with the following piece of code:

**For example**

Consider a class having an array of numbers, and if the number is less than 30, then we will push them into the child queue.

**In ES5, it can be done as follows**

1. **this**.num.forEach(function(num) {
2. **if** (num < 30)
3. **this**.child.push(num);
4. }.bind(**this**));

**In ES6, it can be done by using the arrow function**

1. **this**.num.forEach((num) => {
2. **if** (num < 30)
3. **this**.child.push(num);
4. });