Functions

A Function is a block of code designed to perform a task and executed when it is called or invoked.

There are 3 ways of writing a function in JavaScript:

1. Function Declaration
2. Function Expression
3. Arrow Function

**Syntax**

**Function Declaration (Traditional Function)**

  JS

// Function declaration

function add(a, b) {

console.log(a + b);

}

// Calling a function

add(2, 3);

**Function Expression (Anonymous Function)**

 // Function Expression

JS

  const add = function(a, b) {

        console.log(a+b);

    }

    // Calling function

    add(2, 3);

**Arrow Functions:**

Arrow functions are been introduced in the ES6 version of JavaScript. It is used to shorten the code. Here we do not use the “function” keyword and use the arrow symbol.

JS

  // Single line of code

    let add = (a, b) => a + b;

    console.log(add(3, 2));

JS

// Multiple line of code

    const great = (a, b) => {

        if (a > b)

            return "a is greater";

        else

            return "b is greater";

    }

    console.log(great(3,5));

An arrow function expression is a compact alternative to a traditional function expression, with some semantic differences and deliberate limitations in usage:

1. Arrow functions don't have their own bindings to this, arguments, or super, and should not be used as methods.
2. Arrow functions cannot be used as constructors. Calling them with new throws a TypeError. They also don't have access to the new.target keyword.
3. Arrow functions cannot use yield within their body and cannot be created as generator functions.

JS

// Array

const materials = [

'Hydrogen',

'Helium',

'Lithium',

'Beryllium'

];

// accessing each elements in an array

console.log(materials.map(material => material.length));

// Expected output: Array [8, 6, 7, 9]

[Syntax](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions#syntax)

JS

() => expression

Pa ram => expression

(param) => expression

(param1, paramN) => expression

() => {

statements

}

param => {

statements

}

(param1, paramN) => {

statements

}

(a, b, ...r) => expression // normal parameter

(a = 400, b = 20, c) => expression // values parameter

([a, b] = [10, 20]) => expression // arrays parameter

({ a, b } = { a: 10, b: 20 }) => expression // object parameter

Arrow functions can be async by prefixing the expression with the async keyword.

JS

async param => expression

async (param1, param2, ...paramN) => {

statements

}

**Examples**

// Traditional anonymous function

(function (a) {

return a + 100;

});

// 1. Remove the word "function" and place arrow between the argument and opening body bracket

(a) => {

return a + 100;

};

// 2. Remove the body braces and word "return" — the return is implied.

(a) => a + 100;

// 3. Remove the parameter parentheses

a => a + 100;

In the example above, both the parentheses around the parameter and the braces around the function body may be omitted. However, they can only be omitted in certain cases.

The parentheses can only be omitted if the function has a single simple parameter. If it has multiple parameters, no parameters, or default, destructured, or rest parameters, the parentheses around the parameter list are required.

JS

// Traditional anonymous function

(function (a, b) {

return a + b + 100;

});

// Arrow function

(a, b) => a + b + 100;

const a = 4;

const b = 2;

// Traditional anonymous function (no parameters)

(function () {

return a + b + 100;

});

// Arrow function (no parameters)

() => a + b + 100;

The braces can only be omitted if the function directly returns an expression. If the body has additional lines of processing, the braces are required — and so is the return keyword. Arrow functions cannot guess what or when you want to return.

JS

// Traditional anonymous function

(function (a, b) {

const chuck = 42;

return a + b + chuck;

});

// Arrow function

(a, b) => {

const chuck = 42;

return a + b + chuck;

};

Arrow functions are always unnamed. If the arrow function needs to call itself, use a named function expression instead. You can also assign the arrow function to a variable so it has a name.

JS

// Traditional Function

function bob(a) {

return a + 100;

}

// Arrow Function

const bob2 = (a) => a + 100;

### [Function body](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions#function_body)

Arrow functions can have either a concise body or the usual block body.

In a concise body, only a single expression is specified, which becomes the implicit return value. In a block body, you must use an explicit return statement.

JS

const func = (x) => x \* x;

// concise body syntax, implied "return"

const func2 = (x, y) => {

return x + y;

};

// with block body, explicit "return" needed

Returning object literals using the concise body syntax (params) => { object: literal } does not work as expected.