<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions>

more reference –

<https://hacks.mozilla.org/2015/06/es6-in-depth-arrow-functions/>

An **arrow function expression** is a syntactically compact alternative to a regular [function expression](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/function), although without its own bindings to the [this](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/this), [arguments](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/arguments), [super](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/super), or [new.target](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/new.target) keywords. Arrow function expressions are ill suited as methods, and they cannot be used as constructors.

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

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

(param1, param2, …, paramN) => { statements }

(param1, param2, …, paramN) => expression

// equivalent to: => { return expression; }

// Parentheses are optional when there's only one parameter name:

(singleParam) => { statements }

singleParam => { statements }

// The parameter list for a function with no parameters should be written with a pair of parentheses.

() => { statements }

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

// Parenthesize the body of function to return an object literal expression:

params => ({foo: bar})

// [Rest parameters](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/rest_parameters) and [default parameters](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Default_parameters) are supported

(param1, param2, ...rest) => { statements }

(param1 = defaultValue1, param2, …, paramN = defaultValueN) => {

statements }

// [Destructuring](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment) within the parameter list is also supported

var f = ([a, b] = [1, 2], {x: c} = {x: a + b}) => a + b + c;

f(); // 6

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

See also ["ES6 In Depth: Arrow functions" on hacks.mozilla.org](https://hacks.mozilla.org/2015/06/es6-in-depth-arrow-functions/).

Two factors influenced the introduction of arrow functions: shorter functions and no existence of this keyword.

Shorter functions[Section](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions#Shorter_functions)

var elements = [

'Hydrogen',

'Helium',

'Lithium',

'Beryllium'

];

elements.map(function(element) {

return element.length;

}); // this statement returns the array: [8, 6, 7, 9]

// The regular function above can be written as the arrow function below

elements.map((element) => {

return element.length;

}); // [8, 6, 7, 9]

// When there is only one parameter, we can remove the surrounding parenthesies:

elements.map(element => {

return element.length;

}); // [8, 6, 7, 9]

// When the only statement in an arrow function is `return`, we can remove `return` and remove

// the surrounding curly brackets

elements.map(element => element.length); // [8, 6, 7, 9]

// In this case, because we only need the length property, we can use destructuring parameter:

// Notice that the string `"length"` corresponds to the property we want to get whereas the

// obviously non-special `lengthFooBArX` is just the name of a variable which can be changed

// to any valid variable name you want

elements.map(({ "length": lengthFooBArX }) => lengthFooBArX); // [8, 6, 7, 9]

// This destructuring parameter assignment can be written as seen below. However, note that there

// is no specific `"length"` to select which property we want to get. Instead, the literal name

// itself of the variable `length` is used as the property we want to retrieve from the object.

elements.map(({ length }) => length); // [8, 6, 7, 9]

No separate this[Section](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions#No_separate_this)

Until arrow functions, every new function defined its own [this](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/this) value based on how the function was called:

* A new object in the case of a constructor.
* undefined in [strict mode](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Strict_mode) function calls.
* The base object if the function is called as an "object method".
* etc.

This proved to be less than ideal with an object-oriented style of programming.

function Person() {

// The Person() constructor defines `this` as an instance of itself.

this.age = 0;

setInterval(function growUp() {

// In non-strict mode, the growUp() function defines `this`

// as the global object (because it's where growUp() is executed.),

// which is different from the `this`

// defined by the Person() constructor.

this.age++;

}, 1000);

}

var p = new Person();

In ECMAScript 3/5, the this issue was fixable by assigning the value in this to a variable that could be closed over.

function Person() {

var that = this;

that.age = 0;

setInterval(function growUp() {

// The callback refers to the `that` variable of which

// the value is the expected object.

that.age++;

}, 1000);

}

Alternatively, a [bound function](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/bind) could be created so that a preassigned this value would be passed to the bound target function (the growUp() function in the example above).

An arrow function does not have its own this. The this value of the enclosing lexical scope is used; arrow functions follow the normal variable lookup rules. So while searching for this which is not present in current scope they end up finding this from its enclosing scope.

Thus, in the following code, the this within the function that is passed to setInterval has the same value as this in the lexically enclosing function:

function Person(){

this.age = 0;

setInterval(() => {

this.age++; // |this| properly refers to the Person object

}, 1000);

}

var p = new Person();

**Relation with strict mode**

Given that this comes from the surrounding lexical context, [strict mode](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Strict_mode) rules with regard to this are ignored.

var f = () => { 'use strict'; return this; };

f() === window; // or the global object

All other strict mode rules apply normally.

**Invoked through call or apply**

Since arrow functions do not have their own this, the methods call() or apply() can only pass in parameters. thisArg is ignored.

var adder = {

base: 1,

add: function(a) {

var f = v => v + this.base;

return f(a);

},

addThruCall: function(a) {

var f = v => v + this.base;

var b = {

base: 2

};

return f.call(b, a);

}

};

console.log(adder.add(1)); // This would log to 2

console.log(adder.addThruCall(1)); // This would log to 2 still

No binding of arguments[Section](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions#No_binding_of_arguments)

Arrow functions do not have their own [arguments object](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/arguments). Thus, in this example, arguments is simply a reference to the arguments of the enclosing scope:

var arguments = [1, 2, 3];

var arr = () => arguments[0];

arr(); // 1

function foo(n) {

var f = () => arguments[0] + n; // foo's implicit arguments binding. arguments[0] is n

return f();

}

foo(3); // 6

In most cases, using [rest parameters](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/rest_parameters) is a good alternative to using an arguments object.

function foo(n) {

var f = (...args) => args[0] + n;

return f(10);

}

foo(1); // 11

Arrow functions used as methods[Section](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions#Arrow_functions_used_as_methods)

As stated previously, arrow function expressions are best suited for non-method functions. Let's see what happens when we try to use them as methods:

'use strict';

var obj = {

i: 10,

b: () => console.log(this.i, this),

c: function() {

console.log(this.i, this);

}

}

obj.b(); // prints undefined, Window {...} (or the global object)

obj.c(); // prints 10, Object {...}

Arrow functions do not have their own this. Another example involving [Object.defineProperty()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/defineProperty" \o "The static method Object.defineProperty() defines a new property directly on an object, or modifies an existing property on an object, and returns the object.):

'use strict';

var obj = {

a: 10

};

Object.defineProperty(obj, 'b', {

get: () => {

console.log(this.a, typeof this.a, this); // undefined 'undefined' Window {...} (or the global object)

return this.a + 10; // represents global object 'Window', therefore 'this.a' returns 'undefined'

}

});

Use of the new operator[Section](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions#Use_of_the_new_operator)

Arrow functions cannot be used as constructors and will throw an error when used with new.

var Foo = () => {};

var foo = new Foo(); // TypeError: Foo is not a constructor

Use of prototype property[Section](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions#Use_of_prototype_property)

Arrow functions do not have a prototype property.

var Foo = () => {};

console.log(Foo.prototype); // undefined

Use of the yield keyword[Section](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions#Use_of_the_yield_keyword)

The [yield](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/yield) keyword may not be used in an arrow function's body (except when permitted within functions further nested within it). As a consequence, arrow functions cannot be used as generators.

**Function body**[**Section**](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 an expression is specified, which becomes the implicit return value. In a block body, you must use an explicit return statement.

var func = x => x \* x;

// concise body syntax, implied "return"

var func = (x, y) => { return x + y; };

// with block body, explicit "return" needed

**Returning object literals**[**Section**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions#Returning_object_literals)

Keep in mind that returning object literals using the concise body syntax params => {object:literal} will not work as expected.

var func = () => { foo: 1 };

// Calling func() returns undefined!

var func = () => { foo: function() {} };

// SyntaxError: function statement requires a name

This is because the code inside braces ({}) is parsed as a sequence of statements (i.e. foo is treated like a label, not a key in an object literal).

Remember to wrap the object literal in parentheses.

var func = () => ({ foo: 1 });

**Line breaks**[**Section**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions#Line_breaks)

An arrow function cannot contain a line break between its parameters and its arrow.

var func = (a, b, c)

=> 1;

// SyntaxError: expected expression, got '=>'

However, this can be ammended by using parenthesies or putting the line break inside the arguments as seen below to ensure that the code stays pretty and fluffy.

var func = (

a,

b,

c

) => (

1

);

// no SyntaxError thrown

**Parsing order**[**Section**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions#Parsing_order)

Although the arrow in an arrow function is not an operator, arrow functions have special parsing rules that interact differently with [operator precedence](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Operator_Precedence) compared to regular functions.

let callback;

callback = callback || function() {}; // ok

callback = callback || () => {};

// SyntaxError: invalid arrow-function arguments

callback = callback || (() => {}); // ok

**More examples**[**Section**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions#More_examples)

// An empty arrow function returns undefined

let empty = () => {};

(() => 'foobar')();

// Returns "foobar"

// (this is an Immediately Invoked Function Expression

// see 'IIFE' in glossary)

var simple = a => a > 15 ? 15 : a;

simple(16); // 15

simple(10); // 10

let max = (a, b) => a > b ? a : b;

// Easy array filtering, mapping, ...

var arr = [5, 6, 13, 0, 1, 18, 23];

var sum = arr.reduce((a, b) => a + b);

// 66

var even = arr.filter(v => v % 2 == 0);

// [6, 0, 18]

var double = arr.map(v => v \* 2);

// [10, 12, 26, 0, 2, 36, 46]

// More concise promise chains

promise.then(a => {

// ...

}).then(b => {

// ...

});

// Parameterless arrow functions that are visually easier to parse

setTimeout( () => {

console.log('I happen sooner');

setTimeout( () => {

// deeper code

console.log('I happen later');

}, 1);

}, 1);

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

|  |  |  |
| --- | --- | --- |
| **Specification** | **Status** | **Comment** |
| [ECMAScript 2015 (6th Edition, ECMA-262) The definition of 'Arrow Function Definitions' in that specification.](https://www.ecma-international.org/ecma-262/6.0/#sec-arrow-function-definitions) | Standard | Initial definition. |
| [ECMAScript Latest Draft (ECMA-262) The definition of 'Arrow Function Definitions' in that specification.](https://tc39.github.io/ecma262/#sec-arrow-function-definitions) | Draft |  |

**Browser compatibility**[**Section**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions#Browser_compatibility)

[Update compatibility data on GitHub](https://github.com/mdn/browser-compat-data)

|  | **Desktop** | | | | | | **Mobile** | | | | | | | **Server** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Chrome** | **Edge** | **Firefox** | **Internet Explorer** | **Opera** | **Safari** | **Android webview** | **Chrome for Android** | **Edge Mobile** | **Firefox for Android** | **Opera for Android** | **Safari on iOS** | **Samsung Internet** | **Node.js** |
| **Arrow functions** | Full support45 | Full supportYes | Full support22  Notes  Open | No supportNo | Full support32 | Full support10 | Full support45 | Full support45 | Full supportYes | Full support22  Notes  Open | Full support32 | Full support10 | Full support5.0 | Full supportYes |
| **Trailing comma in parameters** | Full support58 | ? | Full support52 | No supportNo | Full support45 | ? | Full support58 | Full support58 | ? | Full support52 | Full support45 | ? | Full support7.0 | Full supportYes |

Legend

**Full support**

Full support

**No support**

No support

**Compatibility unknown**

Compatibility unknown

**See implementation notes.**

See implementation notes.