Functions

# Array function expressions

An [arrow function](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions) expression is a compact alternative to a traditional function expression, but is limited and can't be used in all situations.

There are differences between arrow functions and traditional functions, as well as some limitations:

1. Arrow functions don't have their own bindings to [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) or [super](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/super), and should not be used as [methods](https://developer.mozilla.org/en-US/docs/Glossary/Method).
2. Arrow functions don't have access to the [new.target](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/new.target) keyword.
3. Arrow functions aren't suitable for [call](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/call), [apply](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/apply) and [bind](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/bind) methods, which generally rely on establishing a [scope](https://developer.mozilla.org/en-US/docs/Glossary/Scope).
4. Arrow functions cannot be used as [constructors](https://developer.mozilla.org/en-US/docs/Glossary/Constructor).
5. Arrow functions cannot use [yield](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/yield), within its body.

**Basic Syntax**

* One param. With simple expression return is not needed: param => expression
* Multiple params require parentheses. With simple expression return is not needed: (param1, paramN) => expression
* Multiline statements require body braces and return: param => {let a = 1; return a + param;}
* Multiple params require parentheses. Multiline statements require body braces and return: (param1, paramN) => { let a = 1; return a + param1 + paramN; }

**Advanced syntax**

* To return an object literal expression requires parentheses around expression: params => ({foo: "a"}) // returning the object {foo: "a"}
* [Rest parameters](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/rest_parameters) are supported: (a, b, ...r) => expression
* [Default parameters](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Default_parameters) are supported: (a=400, b=20, c) => expression
* [Destructuring](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment) within params supported:([a, b] = [10, 20]) => a + b; // result is 30 ({ a, b } = { a: 10, b: 20 }) => a + b; // result is 30

# Argument object

[arguments](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/arguments) is an Array-like object accessible inside functions that contains the values of the arguments passed to that function.

# Getter

The [get](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/get) syntax binds an object property to a function that will be called when that property is looked up.

# setter

The [set](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/set) syntax binds an object property to a function to be called when there is an attempt to set that property.

Promise

The [Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise) object represents the eventual completion (or failure) of an asynchronous operation and its resulting value.

Array Functions

## Reduce

The [reduce()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/reduce) method executes a user-supplied "reducer" callback function on each element of the array, in order, passing in the return value from the calculation on the preceding element. The final result of running the reducer across all elements of the array is a single value.

The first time that the callback is run there is no "return value of the previous calculation". If supplied, an initial value may be used in its place. Otherwise the array element at index 0 is used as the initial value and iteration starts from the next element (index 1 instead of index 0).