**What Are Arrow Functions?**

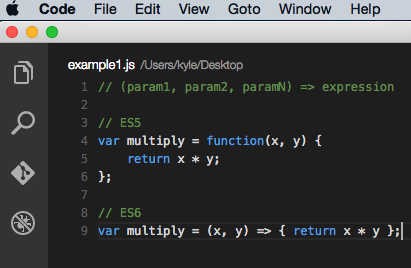
Arrow functions – also called “fat arrow” functions, from CoffeeScript ([a transcompiled language](http://blogs.msdn.com/b/cdnstudents/archive/2013/09/17/visual-studio-tips-for-javascript-coders-try-coffeescript.aspx?WT.mc_id=16547-DEV-sitepoint-article83)) are a more concise syntax for writing function expressions. They utilize a new token, =>, that looks like a fat arrow. Arrow functions are anonymous and change the way this binds in functions.

Arrow functions make our code more concise, and simplify function scoping and the this keyword. They are one-line mini functions which work much like [Lambdas in other languages like C#](https://msdn.microsoft.com/en-us/library/bb397687.aspx?WT.mc_id=16547-DEV-sitepoint-article83) or [Python](http://www.diveintopython.net/power_of_introspection/lambda_functions.html). (See also [lambdas in JavaScript](http://stackoverflow.com/questions/7190439/is-there-a-c-like-lambda-syntax-in-javascript)). By using arrow function we avoid having to type the function keyword, return keyword (it’s implicit in arrow functions), and curly brackets.

**Using Arrow Functions**

There are a variety of syntaxes available in arrow functions. [EcmaScript.org has a thorough list of the syntaxes](http://wiki.ecmascript.org/doku.php?id=harmony:arrow_function_syntax) and [so does MDN](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions). We’ll cover the common ones here to get you started.  
Let’s compare how ES5 code with function expressions can now be written in ES6 using arrow functions.

**Basic Syntax with Multiple Parameters (**[from MDN](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions)**)**



Code Example: <http://codepen.io/DevelopIntelligenceBoulder/pen/wMdPoj?editors=101>

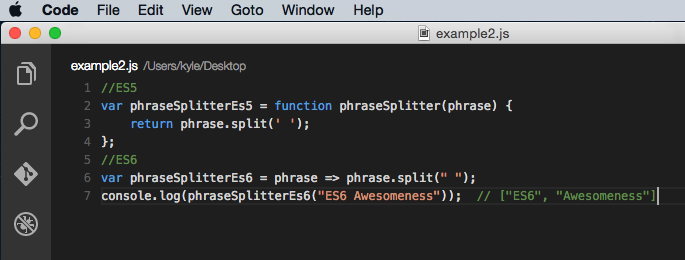
The arrow function example above allows a developer to accomplish the same result with fewer lines of code and approximately half of the typing.

Curly brackets are not required if only one expression is present. The preceding example could also be written as:

var multiply = (x, y) => x\*y;

**Basic Syntax with One Parameter**

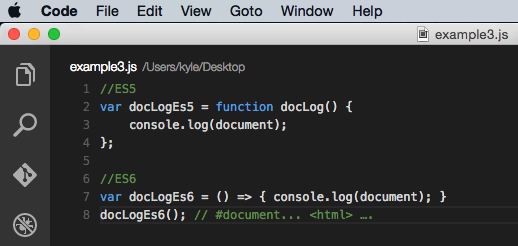
Parentheses are optional when only one parameter is present



<http://codepen.io/DevelopIntelligenceBoulder/pen/PZmOWQ?editors=101>

**No Parameters**

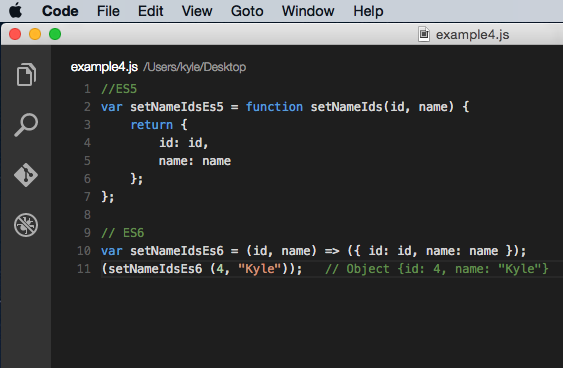
Parentheses are required when no parameters are present.



Code Example: <http://codepen.io/DevelopIntelligenceBoulder/pen/GomOWO?editors=101>

**Object Literal Syntax**

Arrow functions, like function expressions, can be used to return an object literal expression. The only caveat is that the body needs to be wrapped in parentheses, in order to distinguish between a block and an object (both of which use curly brackets).



Code example: <http://codepen.io/DevelopIntelligenceBoulder/pen/zrwPwx?editors=101>

**Use Cases for Arrow Functions**

Now that we’ve covered the basic syntaxes, let’s get into how arrow functions are used.

One common use case for arrow functions is array manipulations and the like. It’s common that you’ll need to map or reduce an array. Take this simple array of objects:

varsmartPhones = [

{ name:'iphone', price:649 },

{ name:'Galaxy S6', price:576 },

{ name:'Galaxy Note 5', price:489 }

];

We could create an array of objects with just the names or prices by doing this in ES5:

// ES5

console.log(smartPhones.map(

function(smartPhone) {

returnsmartPhone.price;

}

)); // [649, 576, 489]

An arrow function is more concise and easier to read:

// ES6

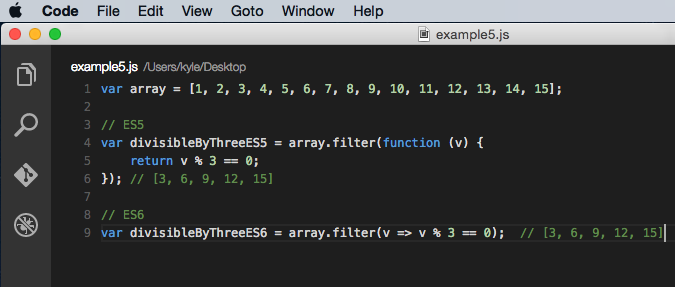
console.log(smartPhones.map(

smartPhone=&gt;smartPhone.price

)); // [649, 576, 489]

Code Example: <http://codepen.io/DevelopIntelligenceBoulder/pen/jWmamX?editors=101>

Here’s another example using the [array filter method](https://msdn.microsoft.com/en-us/library/ff679973(v=vs.94).aspx):



Code Example: <http://codepen.io/DevelopIntelligenceBoulder/pen/RrVjgL?editors=101>

**Promises and Callbacks**

Code that makes use of asynchronous callbacks or promises often contains a great deal of function and return keywords. When using promises, these function expressions will be used for chaining. Here’s a simple example of [chaining promises from the MSDN docs](https://msdn.microsoft.com/en-us/library/windows/apps/hh700334.aspx?WT.mc_id=16547-DEV-sitepoint-article83):

// ES5

aAsync().then(function() {

returnbAsync();

}).then(function() {

returncAsync();

}).done(function() {

finish();

});

This code is simplified, and arguably easier to read using arrow functions:

// ES6

aAsync().then(() =&gt; bAsync()).then(() =&gt; cAsync()).done(() =&gt; finish);

Arrow functions should similarly simplify callback-laden NodeJS code.

***What’s the meaning of this?!***

The other benefit of using arrow functions with promises/callbacks is that it reduces the confusion surrounding the this keyword. In code with multiple nested functions, it can be difficult to keep track of and remember to bind the correct this context. In ES5, you can use workarounds like the .bind method ([which is slow](https://jsperf.com/function-bind-performance/5)) or creating a closure using var self = this;.

Because arrow functions allow you to retain the scope of the caller inside the function, you don’t need to create self = this closures or use bind.

Developer [Jack Franklin](https://twitter.com/jack_franklin) provides an excellent [practical example of using the arrow function lexical this to simplify a promise](http://javascriptplayground.com/blog/2014/04/real-life-es6-arrow-fn/):

Without Arrow functions, the promise code needs to be written something like this:

// ES5

API.prototype.get = function(resource) {

var self = this;

return new Promise(function(resolve, reject) {

http.get(self.uri + resource, function(data) {

resolve(data);

});

});

};

Using an arrow function, the same result can be achieved more concisely and clearly:

// ES6

API.prototype.get = function(resource) {

return new Promise((resolve, reject) =&gt; {

http.get(this.uri + resource, function(data) {

resolve(data);

});

});

};

You can use function expressions if you need a dynamic this and arrow functions for a lexical this.