## Fat Arrow Functions

**Syntax**

JavaScript has *first class functions*.

This means that in JavaScript functions can be themselves be passed around like any other value, even as arguments to other functions.

E.g. we can pass to the setTimeout function, a function, like so:

setTimeout(function() {

console.log("setTimeout called!");

}

, 1000);

The function we pass as an argument to setTimeout is called an *anonymous function* because it doesn’t have a name.

ES6 has introduced a slightly different syntax to define anonymous functions called the *fat arrow* syntax, with it we can re-write the above as:

setTimeout(() => {

console.log("setTimeout called!")

}

, 1000);

If the function only contains one expression we can drop the braces and shorten even further to:

setTimeout(() => console.log("setTimeout called!"), 1000);

**Arguments**

What if we wanted to pass an argument to the function?

We can re-write the below normal function to one that uses the fat arrow syntax:

let add = function(a,b) {

return a + b;

}

;

Can now be written as:

let add = (a,b) => a + b;

In the first example we write return a + b but in the *fat arrow* version we just

wrote a + b. That’s because in the fat arrow syntax if it is on one line, the statement gets returned automatically without the need to use the return keyword.

**this**

Lets imagine we have an object with a function called sayLater, like so:

let obj = {

name: "asim",

sayLater: function() {

console.log(`${this.name}`);

}

}

;

obj.sayLater();

In the

sayLater

function

this

points to

obj

NOTE: In the above the calling context is obj. As the function sayLater is being called in the context of obj, so the function sayLater() checks for the “name” property in the calling context as this keyword is used.

So console.log(${this.name}`);` prints out asim.

Now lets imagine that we log the message using the setTimeout function.

let obj = {

name: "asim",

sayLater: function () {

setTimeout(function () {

console.log(`${this.name}`);

}, 1000);

}

}

;

obj.sayLater();

In the sayLater() function we call setTimeout and then log the value of this.name, which we expect to be *asim*.

In fact we get undefined printed out to the console.

### Calling context

The reason for this is that the value of this in a function depends on *how* the function is called.

At it’s most basic level if the function is called as obj.sayLater(), the value of this is the calling context which in this case is obj.

What is the calling context for the anonymous function we pass to setTimeout? What will this point to inside that function?

setTimeout(function () {

console.log(`${this.name}`);

}

, 1000);

The answer is *it depends*.

In the browser it’s either undefined or the global object depending on if you are running in strict mode or not. In node it’s an internal timeout object.

In all cases however it *isn’t* going to be obj, so this.name is not going to return *asim*, it’s going to return undefined or raise an error.

This *instability* of this is an incredibly common problem in javascript that has affected it since the early days.

In ES5 there are a number of methods we can use to stabilise the value of this. One common solution is to assign this to another variable at the top, usually called self or vm, and then always use self in the function body, like so:

let obj = {

name: "asim",

sayLater: function () {

let self = this; // Assign to self (as now this becomes func. var. – closures will come into picture)

console.log(self);

setTimeout(function () {

console.log(`${self.name}`); // Use self not this

}, 1000);

}

}

;

But in ES6 we can do better, if we use *fat arrow* functions the value of this inside a fat arrow function will be the same as the value of this *outside* the fat arrow function.

It uses the value of this from the surrounding code for its context. i.e. whatever this points to in the surrounding code, this will point to in the function body of the fat arrow function.

We can re-write our obj to use fat arrow syntax like so:

let obj = {

name: "asim",

sayLater: function () {

console.log(this); // `this` points to obj

setTimeout(() => {

console.log(this); // `this` points to obj

console.log(`${this.name}`); // `this` points to obj

}, 1000);

}

}

;

obj.sayLater();

If we ran the above code we would see that the value of this in the setTimout function is now obj; the same as the value of this outside the setTimeout function.

Note – Probably the javascript runtime engine might be using the similar logic as SELF when passing FAT ARROW FUNCTION and for its execution

**Summary**

The new fat arrow function syntax in ES6 is far more than just a slightly shorter way of writing anonymous functions.

It has finally solved the thorny issue of stabilising the value of this that has affected javascript since the start and caused so many work arounds to be discussed and applied in code.

**Listing**

*script.js*

'use strict';

setTimeout(() => {

console.log("setTimeout called!")

}

, 1000);

setTimeout(() => console.log("setTimeout called!"), 1000);

let add = (a,b) => a + b;

console.log(add(1,2));

let obj = {

name: "Asim",

sayLater: function() {

setTimeout(() => console.log(`${this.name}`), 1000)

}

}

;

obj.sayLater();