El uso de JavaScript Arrow Function es cada día más habitual cuando programamos con Javascript . ¿Para qué sirve un arrow function? . En principio muchos de nosotros entendemos los arrows funcions como sintaxis sugar que nos ayuda a simplificar el como trabajamos con JavaScript. Vamos a ver un ejemplo sencillo:

```
function sumar(a,b) {

return a+b;


}

function restar (a,b) {

  return a-b;
}
console.log(sumar(2,2));
console.log(restar(2,2));
```

Acabamos de crearnos dos funciones muy sencillas para hacer dos operaciones básicas .Si ejecutamos el programa por ejemplo desde la consola de node.js el resultado es trivial:

Podemos hacer algo similar utilizando javascript arrow function con lo cual el código queda un poco más compacto y no hace falta declarar funciones . Algo como lo siguiente:

```
var suma=(a,b)=&gt;;a+b;
 var resta=(a,b)=&gt;a-b;

console.log(suma(2,2));
console.log(resta(2,2));

&lt;pre&gt;
```

El resultado es el mismo solamente que usamos una sintaxis un poco más compacta ya que las funciones flecha no han de ser declaradas con la palabra function y no necesitan incluir sintaxis de return.

El resultado es idéntico . Ahora bien el concepto de JavaScript Arrow function aportan más cosas. Supongamos que tenemos el siguiente código:

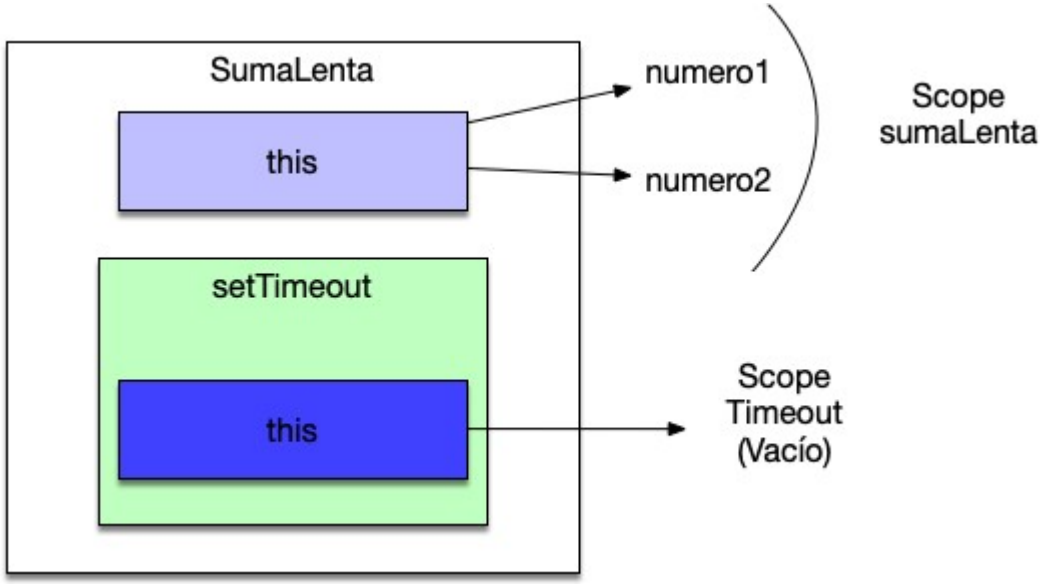
```
function sumar(a,b) {

return a+b;


}

function restar (a,b) {

  return a-b;
}
```

```
function sumaLenta(a,b) {


this.numero1=a;
this.numero2=b;

  setTimeout(function() {

          console.log(sumar(this.numero1,this.numero2));
  },1000)

}

sumaLenta(2,2);
```

En principio parece un código muy sencillo que almacena directamente los valores a, b en el objeto this y luego simplemente invoca el método sumar dentro de un timeout con lo cual tardamos un segundo en ejecutar el código . El mensaje será el mismo pero tardara un segundo en presentar el resultado. Lamentablemente el resultado es diferente.

Javascript Arrow Function y scopes

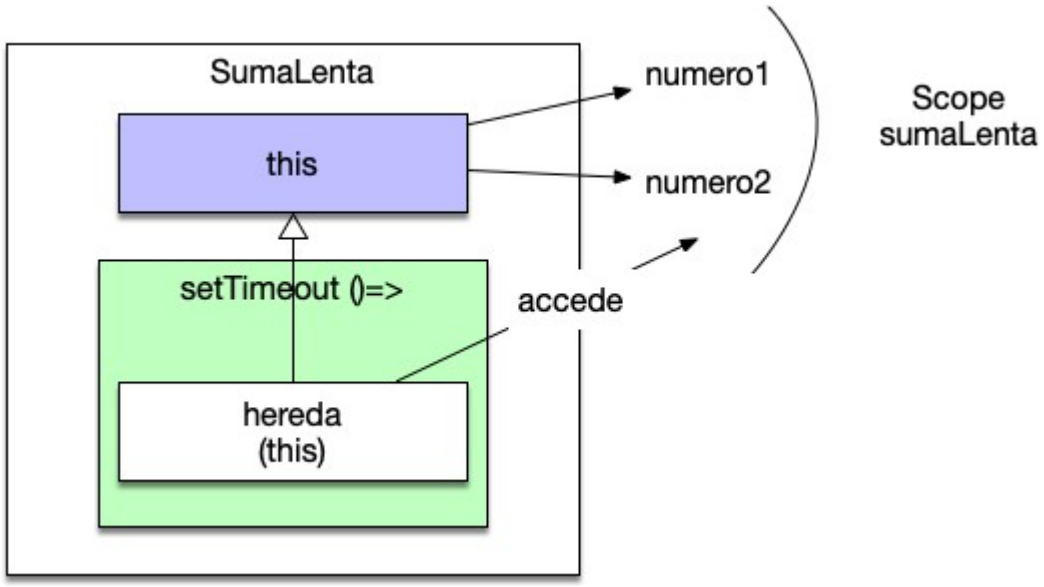
¿Qué es lo que esta pasando? . Lo que sucede es que la función setTimeout recibe como parámetro otra función y por lo tanto genera un nuevo scope o ámbito en JavaScript.



Esto hace que el operador this apunte a un nuevo scope vacío puede parecer algo trivial pero siempre genera problemas y quebraderos de cabeza. ¿Cómo podemos solventar esto? . Muy sencillo podemos cambiar la función que tenemos declarada en el setTimeout y substituirla por una JavaScript Arrow Function que no genera nuevos scopes.

```
function sumaLenta(a,b) {


this.numero1=a;
this.numero2=b;

  setTimeout(()=> {

          console.log(sumar(this.numero1,this.numero2));
  },1000)

}
```

Las arrow function heredan el scope de las funciones padres que las almacenan y por lo tanto podremos acceder sin problema a los números y el programa volverá a funcionar.



El resultado en pantalla realiza la suma:

```
4
```

Siempre que podamos optemos por usar arrow functions ya que evitan la generación de ámbitos adicionales y simplifican las cosas.

1. JavaScript Destructuring ,un concepto interesante
2. JavaScript Template for loop y como usarlo
3. JavaScript Array Spread Operator y simplificaciones
4. JavaScript reduce y su flexibilidad
5. http://es6-features.org/