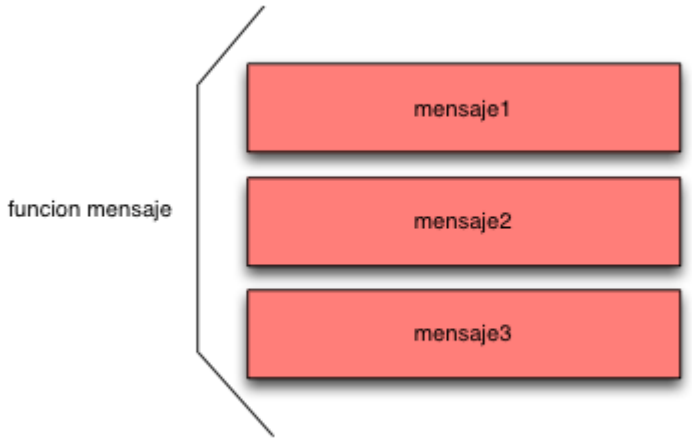
¿Para que sirven los JavaScript Generators? . Los JavaScript Generators son parte de JavaScript ES6 y sirven para realizar una gestión asíncrona de nuestro código mucho más controlada , algo que hasta este momento no era tan sencillo de realizar con el JavaScript clásico. Para entender este concepto vamos a construir un ejemplo sencillo de una función que nos imprima varios mensajes en la consola



Vamos a ver su código:

```
<html>
<head>
<script type="text/javascript">

function mensajes() {

console.log("mensaje1");
console.log("mensaje2");
console.log("mensaje3");
```

```
}

mensajes();

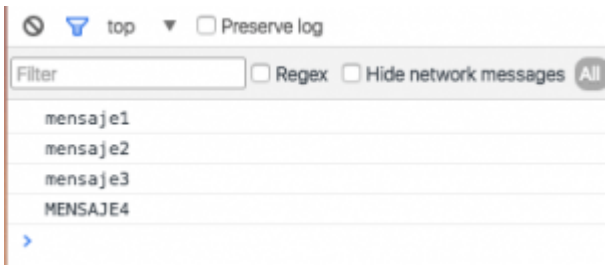
console.log("MENSAJE4");

</script>

</head>
<body>

</body>
</html>
```

Si ejecutamos el resultado aparecerá en la consola



Todo es muy sencillo, si quisiéramos realizar un cambio elemental y que el mensaje3 apareciera después del MENSAJE4 las cosas se complicarían enormemente. Deberíamos realizar alguna operación tipo setTimeout o desmontar la función mensajes. Este tipo de casuísticas se pueden gestionar mucho mejor a través de Generators.

Un ejemplo de Javascript Generators

Los Generators son funciones que devuelven iteradores y permiten un control mucho más fino del código asíncrono. Vamos a modificar la función para trabajar con Generators:

```
<html>
<head>
<script type="text/javascript">

function* mensajes() {

console.log("mensaje1");
console.log("mensaje2");
yield null;
console.log("mensaje3");

}

var iterador = mensajes();
iterador.next();
console.log("MENSAJE4");

iterador.next();
</script>

</head>
<body>

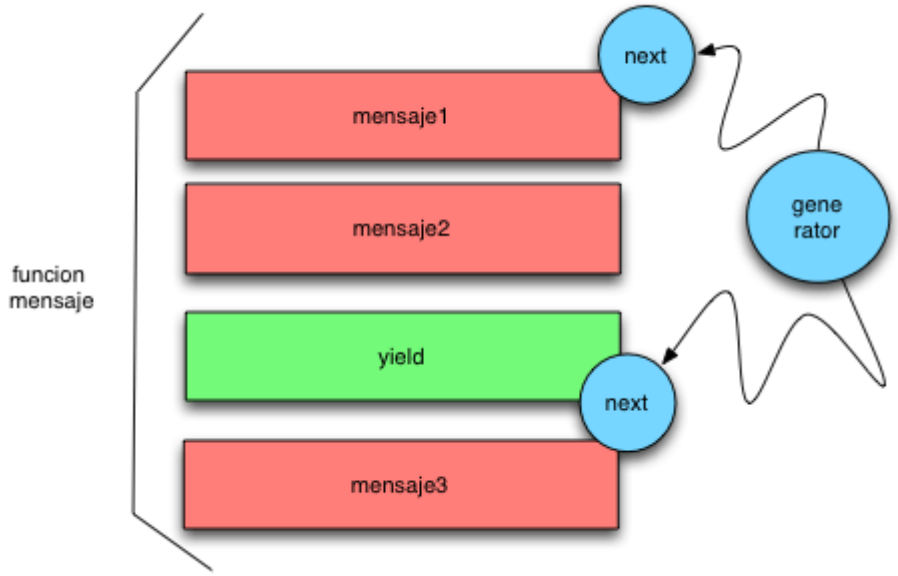
</body>
```

```
</html>
```

Como podemos ver el código es similar pero ahora tenemos una función que tiene un * en su definición y es capaz de trabajar de una forma mucho más asíncrona. Si revisamos el código veremos que aparece una nueva línea

yield null;

Esta linea se encarga de parar la ejecución del programa y “ceder” el control al programa principal . El programa principal recibe un iterador que podemos recorrer según nos convenga.



En este caso ejecutamos la primera iteración e imprimimos mensaje1 y mensaje2 . A continuación ejecutamos el código que nos interesa “MENSAJE4” y finalmente volvemos a invocar al iterador para que realice la siguiente operación imprimir el mensaje3 . Con el uso

de Generators podremos conseguir aumentar la flexibilidad de nuestro código a la hora de realizar tareas asíncronas.

Otros artículos relacionados: Streams vs Promises , JavaScript Promises