Ejercicio Entregable 4

Introducción a las funciones asíncronas

El ejercicio de esta hoja es entregable y puede realizarse en el laboratorio. Los ejercicios entregables realizados a lo largo del curso computan el 10 % de la nota final, pero su entrega no es obligatoria para aprobar la asignatura. La entrega se realiza por parejas.

Fecha límite de entrega: 9 de noviembre de 2017.

La entrega se realizará a través del Campus Virtual.

Aunque este ejercicio tiene una temática algo alejada de los ejercicios que hemos visto hasta ahora, es fundamental para comprender el modelo de programación que utilizaremos a lo largo del curso. Partimos de la siguiente clase:

```
class FlujoNumeros {
    constructor(nums = [6, 1, 4, 3, 10, 9, 8]) { ... }

    siguienteNumero(f) { ... }
}
```

Cada instancia de esta clase encapsula una lista de números a la que podremos acceder secuencialmente, desde el primero hasta el último. Cada llamada a `siguienteNumero` nos permite acceder al siguiente número del array, de modo que podemos obtener todos los números del array mediante sucesivas llamadas a `siguienteNumero`. La particularidad de esta función es el *modo* en el que nos proporciona ese número. Al contrario que las funciones a las que estamos habituados, la función `siguienteNumero` **no nos devuelve** el siguiente número. De hecho, no devuelve nada. En su lugar, la función `siguienteNumero` recibe como parámetro una función `f`, y **llama a esa función**, pasándole el siguiente número como argumento.

Por ejemplo, el siguiente código

```
let flujo = new FlujoNumeros();
flujo.siguienteNumero(num => {
    console.log(`He recibido: ${num}`);
});
```

imprime por pantalla la cadena `He recibido: 6`, ya que el 6 es el primer número de la secuencia. Cuando no quedan más elementos por proporcionar en la secuencia, se llamará a la función `f` pasando `undefined` como argumento.

Para hacer las cosas más interesantes, la función `siguienteNumero` introduce un pequeño retardo de 100ms antes de llamar a `f` con el siguiente número de la lista. Pero, ¡ojo!. Esto no significa que la ejecución del programa se detenga durante esos 100ms. Mientras transcurre este retardo se irán ejecutando las instrucciones que vayan después de `siguienteNumero`. Por ejemplo, suponemos el siguiente fragmento:

```
let flujo = new FlujoNumeros();
flujo.siguienteNumero(num => {
    console.log(`He recibido: ${num}`);
});
console.log('Estoy aquí.')
```

La cadena `He recibido: 6` se mostrará en pantalla transcurridos 100ms desde la llamada al método `flujo.siguienteNumero()` pero, mientras tanto, la ejecución del programa continúa por la siguiente sentencia tras esta llamada, de modo que el resultado de ejecutar el programa es el siguiente:

```
Estoy aquí.
He recibido: 6
```

1. Implementa una función `sumaDosLog(flujo)` que obtenga los dos primeros elementos del `flujo` pasado como parámetro e imprima por pantalla su suma. Puedes suponer que existen, al menos, dos elementos en el flujo de entrada. Por ejemplo:

   ```
   sumaDosLog(new FlujoNumeros());
   // Imprime: 7
   ```

2. Implementa una pequeña variación de la función anterior. Esta vez se trata de escribir una función `sumaDos(flujo, f)` que, en lugar de imprimir por pantalla la suma de los dos primeros números del flujo, llame a la función `f` pasando dicha suma como argumento. Por ejemplo:

   ```
   sumaDos(new FlujoNumeros(), suma => {
       console.log(`El resultado de la suma de los dos primeros números es ${suma}`);
   });
   // Imprime: El resultado de la suma de los dos primeros números es 7
   ```

3. Por último, escribe una función `sumaTodo(flujo, f)` que devuelva la suma de *todos* los números del flujo de entrada (no solo los dos primeros), y llame a `f` pasándole el resultado de la suma.

   ```
   sumaTodo(new FlujoNumeros(), suma => {
       console.log(`El resultado de la suma de todos los números es ${suma}`);
   });
   // Imprime: El resultado de la suma de todos los números es 41
   ```

   Indicación: Es más dificil de lo que parece a primera vista. Te recomiendo que definas una función auxiliar recursiva que acumule en uno de sus parámetros la suma obtenida hasta el momento.