Tareas a realizar

Lectura

Lee atentamente los siguientes artículos y sus correspondientes subsecciones en caso de que las tengan:

- https://es.javascript.info/prototypes (leer solamente: no importa si no se entienden todos los conceptos)
- https://es.javascript.info/classes (algo denso, pero interesante para comprender gran parte del código JS que se está generando en la actualidad; de nuevo, no hay que comprenderlo todo al 100%, basta "saber que existe")
- https://es.javascript.info/error-handling
- https://es.javascript.info/modules

Los artículos propuestos son de teoría general. La parte práctica tratará en su mayoría sobre **programación funcional**, concretamente:

- Filter
- Reduce

Fichero de la aplicación

Utilizaremos el fichero llamado gestionPresupuesto.js en la carpeta js del repositorio. A no ser que se indique lo contrario, todo el código que se cree deberá guardarse en este fichero.
Modificación de export
Añade las funciones filtrarGastos y agruparGastos al objeto export del final del fichero.
Define las funciones vacías (sin parámetros y sin cuerpo) en el
fichero gestionPresupuesto.js para que los tests no den error de sintaxis y se puedan ir comprobando conforme se vaya avanzando en la práctica.

Objeto gasto

Métodos

Añade el siguiente método al objeto gasto (en su función constructora crearGasto):

- obtenerPeriodoAgrupacion - Función de **un parámetro** que **devolverá** el **período de agrupación** correspondiente al parámetro periodo de la función y a la fecha del gasto. Si el período a agrupar es dia, el período de agrupación tendrá el formato aaaa-mm-dd; si es mes, tendrá el formato aaaa-mm; y si es anyo, tendrá el formato aaaa. Ejemplos:
- let gasto1 = new CrearGasto("Gasto 1", 23.55, "2021-09-06", "casa", "supermercado");
- let gasto2 = new CrearGasto("Gasto 2", 27.55, "2021-11-24", "casa", "supermercado", "comida");
-
- gasto1.obtenerPeriodoAgrupacion("mes");
- // Resultado: "2021-09"

- gasto1.obtenerPeriodoAgrupacion("anyo");
- // Resultado: "2021"
- gasto1.obtenerPeriodoAgrupacion("dia");
- // Resultado: "2021-09-06"
-
- gasto2.obtenerPeriodoAgrupacion("mes");
- // Resultado: "2021-11"
- gasto2.obtenerPeriodoAgrupacion("anyo");
- // Resultado: "2021"
- gasto2.obtenerPeriodoAgrupacion("dia");
- // Resultado: "2021-11-24"

```
//Método obtenerPeriodoAgrupación
this.obtenerPeriodoAgrupacion = function (periodo){
    let fecha = new Date (this.fecha).toISOString();

    switch (periodo) {
        case "dia":
            return fecha.substr(0,10);
            break;

        case "mes":
            return fecha.substr(0,7);
            break;

        case "anyo":
            return fecha.substr(0,4);
            break;
    }

}
```

```
Función CrearGasto y funcionamiento de objeto gasto
  ✓ Método 'obtenerPeriodoAgrupacion' del objeto gasto

Función filtrarGastos
  1) Filtra los gastos (variable global 'gastos') de acuerdo con los parámetros pasados a la función

Función agruparGastos
  2) Realiza una agrupación de gastos por períodos temporales


1 passing (14ms)
2 failing
```

```
PS C:\Users\rabic\OneDrive\Escritorio\Curso-github\practica_dwec_gestor_presupuesto\js> git add .\gestionPresupuesto.js
PS C:\Users\rabic\OneDrive\Escritorio\Curso-github\practica_dwec_gestor_presupuesto\js> git commit -m "Método obtener período de agrupación"
[master 1ebb6d8] Método obtener período de agrupación
 1 file changed, 28 insertions(+)
```

Funciones

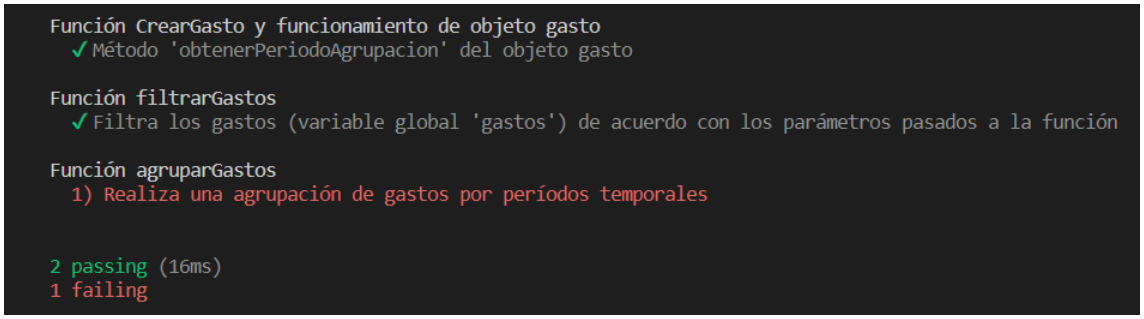
Función filtrarGastos
Función de **un parámetro** que devolverá un subconjunto de los gastos existentes (variable global gastos). Se deberá utilizar la función filter. El parámetro será un **objeto** que podrá tener las siguientes propiedades:

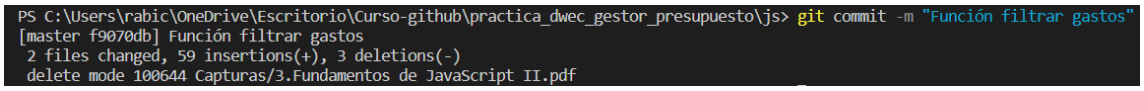
- fechaDesde - Fecha mínima de creación del gasto. Su valor deberá ser un string con formato válido que pueda entender la función Date.parse.
- fechaHasta - Fecha máxima de creación del gasto. Su valor deberá ser un string con formato válido que pueda entender la función Date.parse.
- valorMinimo - Valor mínimo del gasto.
- valorMaximo - Valor máximo del gasto.
- descripcionContiene - **Trozo de texto** que deberá aparecer en la descripción. Deberá hacerse la comparación de manera que no se distingan mayúsculas de minúsculas.
- etiquetasTiene - **Array de etiquetas**: si un gasto contiene **alguna de las etiquetas** indicadas en este parámetro, se deberá devolver en el resultado. Deberá hacerse la comparación de manera que no se distingan mayúsculas de minúsculas.

Algunos ejemplos de llamadas de función filtrarGastos podrían ser:
filtrarGastos({});
filtrarGastos({fechaDesde: "2021-10-10"});
filtrarGastos({fechaDesde: "2021-10-10", fechaHasta: "2021-10-15"});
filtrarGastos({valorMinimo: 10});
filtrarGastos({valorMinimo: 10, valorMaximo: 50});
filtrarGastos({fechaDesde: "2021-10-10", fechaHasta: "2021-10-15", valorMaximo: 100});
filtrarGastos({descripcionContiene: "carne", valorMinimo: 10, valorMaximo: 50});
filtrarGastos({valorMaximo: 50, etiquetasTiene: ["alimentacion"]});
filtrarGastos({etiquetasTiene: ["alimentacion", "gasolina"]});
filtrarGastos({etiquetasTiene: ["alimentacion", "gasolina"], fechaDesde: "2021-10-10"});
filtrarGastos({etiquetasTiene: ["alimentacion", "gasolina"], fechaHasta: "2020-12-31", valorMaximo: 200});

```
//Función filtrarGastos
function filtrarGastos( {fechaDesde, fechaHasta, valorMinimo, valorMaximo, descripcionContiene, etiquetasTiene} ){

    return gastos.filter( function( gasto ) {

        let resultado = true;

        if ( fechaDesde ){
            if( gasto.fecha < Date.parse( fechaDesde )){
                resultado = false;
            }
        }

        if ( fechaHasta ){
            if( gasto.fecha > Date.parse( fechaHasta )){
                resultado = false;
            }
        }

        if ( valorMinimo ){
            if( gasto.valor < valorMinimo ){
                resultado = false;
            }
        }

        if ( valorMaximo ){
            if( gasto.valor > valorMaximo ){
                resultado = false;
            }
        }

        if (descripcionContiene){
            if( gasto.descripcion.indexOf( descripcionContiene ) == -1){
                resultado = false;
            }
        }
```

```
        if (etiquetasTiene){
            let encontrado = false;

            for ( let a of gasto.etiquetas ){
                for ( let b of etiquetasTiene ){

                    if ( a == b ){
                        encontrado = true;
                    }
                }
            }
            if ( !encontrado ){
                resultado = false;
            }
        }

        return resultado;
    });
}
```





Función agruparGastos
Función de **cuatro parámetros** que devolverá **un objeto** con los resultados de realizar una agrupación por período temporal. Los parámetros son:

- *periodo* - Período utilizado para hacer la agrupación. Podrá ser uno de estos tres valores: dia, mes y anyo. El valor por defecto será mes.
- *etiquetas* - **Array** de etiquetas. Solo se seleccionarán los gastos que contengan alguna de esas etiquetas. Si no se indica o es un array vacío, se considerarán todos los gastos.
- *fechaDesde* - Fecha mínima de creación del gasto. Su valor deberá ser un string con formato válido que pueda entender la función Date.parse. Si no se indica se considerarán todos los gastos independientemente de su fecha.
- *fechaHasta* - Fecha máxima de creación del gasto. Su valor deberá ser un string con formato válido que pueda entender la función Date.parse. Si no se indica se considerará la **fecha actual**.

La función realizará los siguientes **pasos**:

1. En primer lugar se llamará a filtrarGastos para obtener el subconjunto de gastos creados entre las fechas indicadas y que tengan alguna de las etiquetas proporcionadas en el parámetro correspondiente.
2. Ejecutar reduce sobre el conjunto de gastos filtrados. El **valor inicial del acumulador** de *reduce* será un **objeto vacío**. Dentro del cuerpo de la función de reduce, **para cada gasto** se obtendrá su **período de agrupación** (a través del método *obtenerPeriodoAgrupacion* del gasto y el parámetro *periodo*), que se utilizará para **identificar la propiedad del acumulador sobre la que se sumará su valor**. Así, si *periodo* = mes, un gasto con fecha 2021-11-01 tendrá un período de agrupación 2021-11, por lo que su valor se sumará a acc["2021-11"] (siempre que la variable del acumulador haya recibido el nombre acc en la llamada a reduce). Tienes una pista sobre cómo proceder en la siguiente pregunta de Stack Overflow.
3. El resultado de reduce será el valor de vuelta de la función agruparGastos.

Algunos ejemplos de resultados de ejecución de agruparGastos:
gastos = [
{id: 0, descripcion: "Gasto 1", valor: 5, fecha: "2021-09-30", etiquetas: ["alimentacion"]},
{id: 1, descripcion: "Gasto 1", valor: 10, fecha: "2021-10-01", etiquetas: ["alimentacion"]},
{id: 2, descripcion: "Gasto 1", valor: 12, fecha: "2021-10-02", etiquetas: ["transporte"]},

```
   {id: 3, descripcion: "Gasto 1", valor: 17, fecha: "2021-10-02", etiquetas: ["alimentacion"]}
];

// Suponemos fecha actual 2021-10-15

let agrup1 = agruparGastos("mes");
/*agrup1 = {
   "2021-09": 5,
   "2021-10": 39
}*/

let agrup2 = agruparGastos("dia");
/*agrup2 = {
   "2021-09-30": 5,
   "2021-10-01": 10,
   "2021-10-02": 29
}*/

let agrup3 = agruparGastos("mes", ["alimentacion"]);
/*agrup3 = {
   "2021-09": 5,
   "2021-10": 27
}*/
```

```
//Función agruparGastos
function agruparGastos (periodo, etiquetas, fechaDesde, fechaHasta){

    return filtrarGastos ({ etiquetasTiene: etiquetas, fechaDesde: fechaDesde, fechaHasta: fechaHasta })

    .reduce (function (acc, gasto) {
        let grupo = gasto.obtenerPeriodoAgrupacion (periodo);

        acc [grupo] = (acc[grupo] || 0) + gasto.valor;

        return acc;
    },

    {})
}

//https://stackoverflow.com/questions/14446511/most-efficient-method-to-groupby-on-an-array-of-objects
    //function groupBy(data, key){
    //return data
    //.reduce(     (acc, cur) => {
    //acc[cur[key]] = acc[cur[key]] || [];
    // if the key is new, initiate its value to an array, otherwise keep its own array value
    //acc[cur[key]].push(cur);
    //return acc;
    //},
    //[])
    //}
```

```
Función CrearGasto y funcionamiento de objeto gasto
  ✓ Método 'obtenerPeriodoAgrupacion' del objeto gasto

Función filtrarGastos
  ✓ Filtra los gastos (variable global 'gastos') de acuerdo con los parámetros pasados a la función

Función agruparGastos
  ✓ Realiza una agrupación de gastos por períodos temporales


3 passing (16ms)
```

```
PS C:\Users\rabic\OneDrive\Escritorio\Curso-github\practica_dwec_gestor_presupuesto\js> git commit -m "Función agrupar gastos"
[master 5319cc9] Función agrupar gastos
 1 file changed, 26 insertions(+), 1 deletion(-)
```

```
PS C:\Users\rabic\OneDrive\Escritorio\Curso-github\practica_dwec_gestor_presupuesto\js> git push
Enumerating objects: 17, done.
Counting objects: 100% (17/17), done.
Delta compression using up to 4 threads
Compressing objects: 100% (13/13), done.
Writing objects: 100% (13/13), 2.00 KiB | 682.00 KiB/s, done.
Total 13 (delta 10), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (10/10), completed with 4 local objects.
To https://github.com/CintiaCastilloCareglio/practica_dwec_gestor_presupuesto.git
   c6549cb..5319cc9  master -> master
```