Las ganancias semestrales

Ana, contadora de una conocida empresa , tiene registros para representar los balances de cada mes y una lista para guardarlos. Por ejemplo, para el último semestre del año pasado registró los siguientes:

```
//En julio ganó $50, en agosto perdió $12, etc
let balancesUltimoSemestre = [
  { mes: "julio", ganancia: 50 },
  { mes: "agosto", ganancia: -12 },
  { mes: "septiembre", ganancia: 1000 },
  { mes: "octubre", ganancia: 300 },
  { mes:  "noviembre", ganancia: 200 },
  { mes: "diciembre", ganancia: 0 }
];
```

Y nos acaba de preguntar: *"¿puedo saber la ganancia de todo* ***un semestre****?"*

"Obvio, solo tenemos que sumar las ganancias de todos los balances", dijimos, y escribimos el siguiente código:

```
function gananciaSemestre(balances) {
  return balances[0].ganancia + balances[1].ganancia +
         balances[2].ganancia + balances[3].ganancia +
         balances[4].ganancia + balances[5].ganancia;
}
```

"Gracias ", nos dijo Ana, y se fue calcular las ganancias usando la función que le pasamos. Pero un rato más tarde, volvió contándonos que también había registrado los balances del primer trimestre de este año:

```
//En enero la empresa ganó $80, en febrero, $453, en marzo $1000
let balancesPrimerTrimestre = [
  { mes: "enero", ganancia: 80 },
  { mes: "febrero", ganancia: 453 },
  { mes: "marzo", ganancia: 1000 }
];
```

Y nos preguntó: *"¿Podría usar esta función que me dieron para calcular las ganancias del primer trimestre?"*.

> ¿Tiene algún problema la función `gananciaSemestre` que escribimos anteriormente? ¿Funcionará con los balances trimestrales? ¿Y con los cuatrimestrestrales?
> ¡Probala en la consola!

Consola Biblioteca

>

¿Y el resto de las ganancias?

La función `gananciaSemestre` anterior tiene dos problemas :

1. Es muy repetitiva y tediosa de escribir. ¡Tenemos que hacer muchas sumas a mano!
2. No es genérica, como bien dice su nombre, sólo sirve para sumar las ganancias de 6 balances:

- si la lista tiene más de seis balances, sólo suma los primeros;
- si tiene menos, no funciona (¿te acordás cuando te dijimos que si te ibas de índice cosas malas podían ocurrir?)

Lo que nos gustaría es poder sumar las ganancias de todos los balances de una lista, sin importar cuántos haya realmente; queremos una función `gananciaTotal` , que pueda sumar balances de cualquier período de meses: semestres, cuatrimestres, trimestres, etc. ¡Qué difícil!

¡Relajá! Ya tenemos nuestra versión; probala con las siguientes consultas:

```
> gananciaTotal([
      { mes: "enero", ganancia: 2 },
      { mes: "febrero", ganancia: 3 }
  ])
> gananciaTotal([
      { mes: "enero", ganancia: 2 },
      { mes: "febrero", ganancia: 3 },
      { mes: "marzo", ganancia: 1 },
      { mes: "abril", ganancia: 8 },
      { mes: "mayo", ganancia: 8 },
      { mes: "junio", ganancia: -1 }
  ])
> gananciaTotal([])
```

Después seguinos para contarte cómo la hicimos.

>

Todas las ganancias, la ganancia

Ahora que sabemos la función que necesitamos (`gananciaTotal`), razonemos cómo hacerla...

3. Todas las ganancias, la ganancia

Vamos de a poquito : si la lista no tuviera elementos, ¿cuánto debería ser la sumatoria? ¡0!

```
function gananciaTotal0(balancesDeUnPeriodo) {
  let sumatoria = 0;
  return sumatoria;
}
```

¿Y si tuviera exactamente 1 elemento? Sería... 0.... ¿más ese elemento? ¡Exacto!

```
function gananciaTotal1(balancesDeUnPeriodo) {
  let sumatoria = 0;
  sumatoria = sumatoria + balancesDeUnPeriodo[0].ganancia;
  return sumatoria;
}
```

¿Y si tuviera 2 elementos?

```
function gananciaTotal2(balancesDeUnPeriodo) {
  let sumatoria = 0;
  sumatoria = sumatoria + balancesDeUnPeriodo[0].ganancia;
  sumatoria = sumatoria + balancesDeUnPeriodo[1].ganancia;
  return sumatoria;
}
```

¿Y si tuviera 3 elementos?

```
function gananciaTotal3(balancesDeUnPeriodo) {
  let sumatoria = 0;
  sumatoria = sumatoria + balancesDeUnPeriodo[0].ganancia;
  sumatoria = sumatoria + balancesDeUnPeriodo[1].ganancia;
  sumatoria = sumatoria + balancesDeUnPeriodo[2].ganancia;
  return sumatoria;
}
```

¿Empezás a ver un patrón? Tratá de escribir `gananciaTotal4` que funcione para 4 elementos.

Solución | Consola

```
function gananciaTotal4(balancesDeUnPeriodo) {
  let sumatoria = 0;
  sumatoria = sumatoria + balancesDeUnPeriodo[0].ganancia;
  sumatoria = sumatoria + balancesDeUnPeriodo[1].ganancia;
  sumatoria = sumatoria + balancesDeUnPeriodo[2].ganancia;
  sumatoria = sumatoria + balancesDeUnPeriodo[3].ganancia;
  return sumatoria;
}
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

¡Bien hecho!

¿Y si la lista tuviera *cualquier* cantidad de elementos? Si seguimos repitiendo este patrón, veremos que una sumatoria de una lista siempre arranca igual, con `let sumatoria = 0`, y termina igual, devolviendo la variable local sumatoria (`return sumatoria`).

```
function gananciaTotalN(unPeriodo) {
  let sumatoria = 0; // esto siempre está
  //... etc
  return sumatoria; //esto siempre está
}
```

Lo que cambia son las acumulaciones (`sumatoria = sumatoria + ...`); necesitamos una por cada elemento de la lista. Dicho de otra forma, tenemos que *visitar* cada elemento del mismo, sin importar cuántos tenga. Pero, ¿cómo hacerlo? ¿No te suena conocida esta idea de *repetir algo muchas veces*?

Nos visita un viejo amigo

Lo que tenemos que hacer, entonces, es repetir la operación de acumular varias veces, una por cada elemento de la lista. ¡Digamos hola (nuevamente) al `for...of`!

4. Nos visita un viejo amigo

```
function gananciaTotal(balancesDeUnPeriodo) {
  let sumatoria = 0;
  for (let balance of balancesDeUnPeriodo) {
    sumatoria = sumatoria + balance.ganancia;
  }
  return sumatoria;
}
```

Como ves, el `for...of` nos permite visitar y hacer algo con cada elemento de una lista; en este caso, estaremos visitando cada `balance` de `balancesDeUnPeriodo`.

¿Aún no te convenciste? Nuevamente, probá las siguientes expresiones en la consola:

```
> gananciaTotal([])
> gananciaTotal([
    { mes: "noviembre", ganancia: 5 }
  ])
> gananciaTotal([
    { mes: "marzo", ganancia: 8 },
    { mes: "agosto", ganancia: 10 }
  ])
> gananciaTotal([
    { mes: "enero", ganancia: 2 },
    { mes: "febrero", ganancia: 10 },
    { mes: "marzo", ganancia: -20 }
  ])
> gananciaTotal([
    { mes: "enero", ganancia: 2 },
    { mes: "febrero", ganancia: 10 },
    { mes: "marzo", ganancia: -20 },
    { mes: "abril", ganancia: 0 },
    { mes: "mayo", ganancia: 10 }
  ])
```

Consola Biblioteca

>

Cuentas claras

¡Ana tiene nuevos requirimientos! Ahora nos pidió lo siguiente: *"Quiero saber cuántos balances fueron positivos, es decir, aquellos en los que la ganancia fue mayor a cero"*.

> Completá la función `cantidadDeBalancesPositivos`. Si prestás atención, notarás que tiene una estructura similar al problema anterior.

¡Dame una pista!

Solución | Consola

```
function cantidadDeBalancesPositivos(balancesDeUnPeriodo) {
  let cantidad = 0;
  for (let balance of balancesDeUnPeriodo) {
    if (balance.ganancia >0)
    cantidad = cantidad + 1;
  }
  return cantidad;
}
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

`cantidad` es lo que en programación se conoce como *contador*, una variable que se incrementa cada vez que hacemos algo dentro de un `for...of` o solo aquellas veces que se cumpla una condición (como en este caso).

La ganancia promedio

Pasemos al siguiente requerimiento de Ana. Ya podemos calcular una sumatoria de ganancias y también crear contadores, ahora vamos a calcular promedios.

Ana quisiera saber dado un conjunto cualquiera de balances cuál es su `gananciaPromedio` .

```
> gananciaPromedio([
      { mes: "marzo", ganancia: 8 },
      { mes: "agosto", ganancia: 10 }
  ])
9
```

¡Dame una pista!

Solución | Consola

```
1 function gananciaPromedio(balancesDeUnPeriodo) {
2   return gananciaTotal(balancesDeUnPeriodo)/longitud(balancesDeUnPeriodo);
3 }
4
5
6
7
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

¡Perfecto! Vamos a complicar un poco las cosas.

Quién gana, quién pierde

Viendo que podemos hacer todo lo que nos pide, Ana quiere saber la ganancia promedio de los balances positivos.

7. Quién gana, quién pierde

Definí las funciones:

- `gananciaPositiva` , que es la suma de las ganancias de los balances positivos
- `promedioGananciasPositivas` invocando `gananciaPositiva` y `cantidadDeBalancesPositivos` .

¡Dame una pista!

Solución | Biblioteca | Consola

```
function gananciaPositiva(balancesDeUnPeriodo) {
  let sumatoria = 0;
  for (let balance of balancesDeUnPeriodo) {
    if (balance.ganancia >0)
    sumatoria = sumatoria + balance.ganancia;
  }
  return  sumatoria;
}


function promedioGananciasPositivas(lista1) {
  return gananciaPositiva(lista1)/cantidadDeBalancesPositivos(lista1);
}

```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Como podés ver todos los promedios se basan en el mismo principio . Sumar una cantidad determinada elementos y dividir el resultado por esa cantidad. Si quisiéramos realizar una función `promedio` genérica sería algo así:

```
function promedio(listaDeNumeros) {
    return sumatoria(listaDeNumeros) / longitud(listaDeNumeros);
}

function sumatoria(listaDeNumeros) {
  let sumatoria = 0;
  for (let numero of listaDeNumeros) {
    sumatoria = sumatoria + numero;
  }
  return sumatoria;
}
```

¡Pero nosotros no tenemos una lista de números sino de registros! ¿Y entonces?

Soy el mapa, soy el mapa

Lamentablemente no se puede usar la función `promedio` con nuestra lista de registros . Lo que necesitamos es una lista que tenga solo las ganancias de cada balance. Para ello debemos transformar, o *mapear*, cada elemento de la lista.

Completá la función `ganancias` que toma una lista de balances y devuelve una lista que solo posea solo las ganancias de cada uno.

```
> ganancias([
      { mes: "enero", ganancia: 40 },
      { mes: "febrero", ganancia: 12 },
      { mes: "marzo", ganancia: 8}
  ])
[40, 12, 8]
```

¡Dame una pista!

Solución | Consola

```
function ganancias(balancesDeUnPeriodo) {
  let ganancias = [];
  for (let balance of balancesDeUnPeriodo) {
    agregar(ganancias,balance.ganancia);
  }
  return ganancias;
}

```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

¡Excelente! Ahora ya sabemos cómo transformar cada elemento de una lista para obtener una lista nueva . De esta manera podemos usar `promedio` con nuestra lista de balances. Pero, ¿se puede utilizar la función `promedio` solo para los balances positivos?

A filtrar, a filtrar cada cosa en su lugar

Con la programación se puede hacer cualquier cosa, o casi . Ya hicimos una función para poder saber la cantidad de balances positivos (`cantidadDeBalancesPositivos`), ahora vamos a ver cómo podemos hacer para saber cuáles son esos balances.

9. A filtrar, a filtrar cada c

Completá la función `balancesPositivos` que toma los balances de un período y devuelve una lista con aquellos cuya ganancia fue mayor a cero.

¡Dame una pista!

Solución | Consola

```
function balancesPositivos(balancesDeUnPeriodo) {
  let balances = [];
  for (let balance of balancesDeUnPeriodo) {
    if (balance.ganancia > 0)
    agregar(balances,balance);
  }
  return balances;
}

```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

¡Muy bien! Ahora ya sabemos cómo filtrar una lista. En criollo, aprendimos a obtener los elementos de una lista que cumplen una condición determinada. En este caso obtuvimos una nueva lista con los balances que presentaban una ganancia positiva.

Un promedio más positivo

Ahora que tenemos la función `ganancias` y `balancesPositivos` podemos utilizar la función `promedio` genérica para saber cuál es el promedio de ganancia de los balances positivos.

> Definí la función `gananciasDeBalancesPositivos` y luego usala junto a `promedio` para definir `promedioDeBalancesPositivos` .

¡Dame una pista!

Solución | Biblioteca | Consola

```
function gananciasDeBalancesPositivos(balancesDeUnPeriodo){
  let ganancias = [];
  for (let balance of balancesPositivos(balancesDeUnPeriodo)){
    agregar(ganancias,balance.ganancia);
  }
  return ganancias;
}

function promedioDeBalancesPositivos(balancesDeUnPeriodo){
  return promedio(gananciasDeBalancesPositivos(balancesDeUnPeriodo));
}
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Esto es lo máximo

Vamos a conocer una nueva función, `maximo` , que nos permite conocer cuál es el mayor valor en una lista de números. Por ejemplo:

```
> maximo([5, 8, 10, 42, 87, 776])
776
```

Usando esta nueva función, definí la función `maximaGanancia` que nos diga cuál es la ganancia más alta entre los balances de un período de tiempo.

```
> maximaGanancia([
      { mes: "enero", ganancia: 87 },
      { mes: "febrero", ganancia: 12 },
      { mes: "marzo", ganancia: 8}
  ])
87
```

¡Dame una pista!

Solución | Biblioteca | Consola

```
1 function maximaGanancia (balancesDeUnPeriodo){
2   return maximo(ganancias(balancesDeUnPeriodo));
3 }
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Si hay una función para calcular el máximo de una lista también hay una para calcular el mínimo. ¿Te imaginás como se llama?

Como mínimo

Suponemos que adivinaste el nombre. En caso que no, es `minimo` .

Definí la función `minimaGananciaPositiva` que nos diga cuál es la ganancia más baja de todos los balances positivos.

```
> minimaGananciaPositiva([
      { mes: "enero", ganancia: -40 },
      { mes: "febrero", ganancia: 42 },
      { mes: "marzo", ganancia: 8},
      { mes: "abril", ganancia:  -5}
  ])
8
```

¡Dame una pista!

Solución | Biblioteca | Consola

```
1 function minimaGananciaPositiva (balancesDeUnPeriodo){
2   return minimo(gananciasDeBalancesPositivos(balancesDeUnPeriodo));
3 }
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

¡Muy bien! Solo queda un ejercicio por delante.

Los mejores meses del año

¡Vamos a terminar esta lección con todo!

Para eso vamos a hacer las siguientes funciones:

- `meses` , la cual dada una lista con registros devuelve una lista de meses ;
- `afortunados` , que filtra aquellos registros que tuvieron una ganancia mayor a $1000 ;
- `mesesAfortunados` , devuelve aquellos meses que fueron afortunados.

```
> meses([
    { mes: "enero", ganancia: 870 },
    { mes: "febrero", ganancia: 1000 },
    { mes: "marzo", ganancia: 1020 },
    { mes: "abril", ganancia: 2300 },
    { mes: "mayo", ganancia: -10 }
  ])
["enero", "febrero", "marzo", "abril", "mayo"]

> afortunados([
      { mes: "enero", ganancia: 870 },
      { mes: "febrero", ganancia: 1000 },
      { mes: "marzo", ganancia: 1020 },
      { mes: "abril", ganancia: 2300 },
      { mes: "mayo", ganancia: -10 }
    ])
[ { mes: "marzo", ganancia: 1020 }, { mes: "abril", ganancia: 2300 }]

> mesesAfortunados([
    { mes: "enero", ganancia: 870 },
    { mes: "febrero", ganancia: 1000 },
    { mes: "marzo", ganancia: 1020 },
    { mes: "abril", ganancia: 2300 },
    { mes: "mayo", ganancia: -10 }
  ])
["marzo", "abril"]
```

Definí las funciones `meses` , `afortunados` , `mesesAfortunados` .

¡Dame una pista!

Solución | Biblioteca | Consola

```
1  function meses(gananciasDeUnPeriodo){
2    let mes = [];
3    for (let periodo of gananciasDeUnPeriodo)
4    agregar(mes, periodo.mes);
5    return mes;
6  }
7
8
9  function afortunados(gananciasDeUnPeriodo){
10   let afortunados = [];
11   for (let periodo of gananciasDeUnPeriodo){
12     if (periodo.ganancia>1000){
13     agregar(afortunados, periodo);
14     }
15   }
16   return afortunados;
17 }
18
19
20 function mesesAfortunados(gananciasDeUnPeriodo){
```

```
21   return meses(afortunados(gananciasDeUnPeriodo));
22 }
23
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas