TALLER FUNCIONES EN TYPESCRIPT

YESI ESTEBAN PANTOJA CUELLAR

DOCENTE

BRAYAN IGNACIO ARCOS BURBANO

INSTITUTO TECNOLÓGICO DEL PUTUMAYO

TECNOLOGÍA EN DESARROLLO DE SOFTWARE

QUINTO SEMESTRE

PROGRAMACIÓN BACKEND

MOCOA - PUTUMAYO

2023

EJERCICIO 1°.

Para este ejercicio trabajaremos con la siguiente constante “auxNumber” que será un arreglo de tipo numérico en la cuál añadiremos los siguientes valores “[1, 2, 3, 4, 5, 6, 7, 8, 9]” quedando de la siguiente manera =>

```
const auxNumber: number[] = [1, 2, 3, 4, 5, 6, 7, 8, 9];
```

Para ejecutar y poder ver este arreglo usamos el método “console.log()”, en el medio de los paréntesis pondremos nuestra constante “auxNumber” y entre comillas simples ponemos un alias en este caso he optado por ponerle ‘number’, quedando de la siguiente manera =>

```
console.log(auxNumber, 'number');
```

Y a la hora de imprimir en consola nos debería de salir lo siguiente =>

```
[
  1, 2, 3, 4, 5,
  6, 7, 8, 9
] number
```

1) agregar al array por medio de la función push los siguientes valores númericos 10,12,15,16,17.

Para resolver este punto del primer ejercicio tenemos que usar la función push() la cual nos permite agregar nuevos valores a nuestro arreglo dependiendo del tipo que se esté empleando. Por ejemplo, en este caso al ser tipo “Number” sólo podremos agregar valores numéricos, sin embargo, no podremos agregar otros valores cómo tipo “String” (texto) o “boolean” (verdadero o falso).

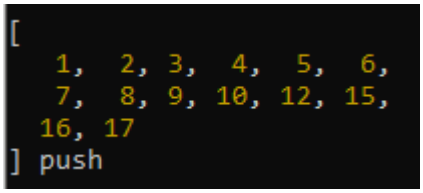
Para lograr esto primero debemos llamar a nuestra variable “auxNumber” y mediante un punto(.) le agregamos la función push(), en medio de los paréntesis vamos a ubicar cada número que se decida agregar. De esta forma debemos hacerlo =>

```
auxNumber.push(10);
auxNumber.push(12);
auxNumber.push(15);
auxNumber.push(16);
auxNumber.push(17);
```

Luego de esto, procedemos a imprimir la nueva estructura de nuestro arreglo “auxNumber” mediante el método “console.log()”, y para este caso le empleamos un alias “push” porque hemos empleado está función. Con lo anterior dicho de esta manera debería quedar nuestro código =>

```
console.log(auxNumber, 'push');
```

Y al ejecutar en consola nos debería dar el siguiente resultado =>



Aquí vemos que los nuevos valores han sido agregados correctamente.

2) Eliminar el último elemento del arreglo mediante la función pop().

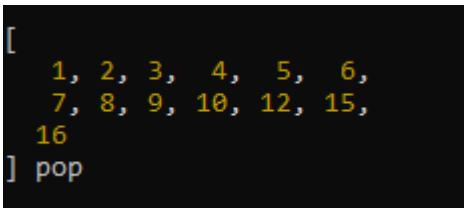
Para hacer este ejercicio empezamos por llamar a nuestra variable “auxNumber” y mediante un punto (.) agregamos la función pop(), ésta eliminará el último valor de nuestro arreglo independientemente de que tipo sea. Nuestro código debería de verse en este estilo =>

```
auxNumber.pop();
```

Luego de esto usamos el método “console.log” para imprimir mediante consola y agregamos un alias en este caso “pop”. Quedándonos de la siguiente manera =>

```
console.log(auxNumber, 'pop');
```

Y a la hora de ejecutar la consola, nos debería salir este resultado =>



Cómo podemos ver el último elemento que antes era 17, ya no aparece, indicando que hicimos bien el ejercicio.

3) Encontrar el índice de los siguientes elementos 1, 4 y 7 mediante la función indexOf().

Para resolver este ejercicio emplearemos la función indexOf(), la cual nos permite mostrar la posición de los valores las cuáles estamos buscando. Para utilizar esta función primero debemos crear tres “constantes” (const) a la cual le asignaremos un nombre cómo los siguientes: “indexAuxNumber1”, “indexAuxNumber4”, “indexAuxNumber7”, luego de esto llamaremos a nuestro arreglo “auxNumber” y mediante un punto (.) empleamos la función indexOf(), entre los paréntesis agregamos nuestros valores. Quedando de la siguiente manera =>

```
const indexAuxNumber1 = auxNumber.indexOf(1);
const indexAuxNumber4 = auxNumber.indexOf(4);
const indexAuxNumber7 = auxNumber.indexOf(7);
```

Y mediante “console.log()” imprimiremos cada constante creada junto a una alias, en este caso “indexOf”. En nuestro código debería de quedar de la siguiente manera =>

```
console.log(indexAuxNumber1, indexAuxNumber4, indexAuxNumber7, 'indexOf');
```

El resultado de esto es el siguiente =>

```
0 3 6 indexOf
```

Esto nos dice que en la posición 0 está el valor del elemento 1, en la posición 3 el valor del elemento 4 y en la posición 6 el elemento 7, hay que aclarar que en los arreglos las posiciones inician por defecto desde 0 en adelante.

4) Extraer del array desde la posición 2 hasta el valor 9 mediante la función slice()

Para este ejercicio como bien se menciona emplearemos la función “slice()” está nos permite traer una parte del arreglo, dónde el primer valor va a ser la posición y el segundo hasta el valor al que queremos llegar. Continuando con nuestro código vamos a declarar una constante en la que almacenaremos el resultado, en este caso lo llamaremos “slicedAuxNumber” que será igual a la variable del arreglo “auxNumber” y con un punto (.) usamos la función slice(), entre los paréntesis agregamos de primero la posición y luego el valor hasta el cual queremos llegar separados por una coma (2, 9). En nuestro código debería de verse de la siguiente manera =>

```
const slicedAuxNumber = auxNumber.slice(2, 9);
```

Luego para mostrar el resultado usaremos console.log como veníamos manejando anteriormente y su alias en este caso será “slice” =>

```
console.log(slicedAuxNumber, 'slice');
```

Ya el resultado obtenido en consola deberá ser el siguiente =>

```
[
  3, 4, 5, 6,
  7, 8, 9
] slice
```

Cómo podemos ver a traído el valor de la posición 2, en este caso el valor es 2, hasta el valor que le pedimos 9.

5) Verificar si todos los elementos son mayores que 1 con every ()

Para verificar esto primero debemos saber que la función every() retorna sólo valores boléanos (verdadero, falso), entonces para hacer esto vamos a declarar una constante llamada “allPositveAuxNumber” a la que le asignaremos un tipo “boolean” y esto será igual a la variable “auxNumber” a la que con un punto (.) agregamos la función every(), entre los paréntesis agregamos un parámetro con el cuál se recorrerá todo el arreglo para poder verificar lo que buscamos, este parámetro lo denominamos “num” al cuál le asignaremos que sí cada numero es mayor a 1, en código se vería así =>

```
const allPositveAuxNumber: boolean = auxNumber.every((num) => num > 1);
```

Ya para mostrar en consola usamos console.log, llamamos la constante con su alias asignado “every” para este caso =>

```
console.log(allPositveAuxNumber, 'every');
```

El resultado debería ser el siguiente =>

```
false every
```

Esto nos quiere decir que no todos los elementos del arreglo son mayores a 1

6) Verificar si algún elemento sea mayor o igual que 7 con la función some()

Esta es otra verificación, pero contraria al anterior en esta se busca es que alguno cumpla el requerimiento. Para hacer este ejercicio primero declaramos una constante llamada “someAuxNumber” al que le asignamos un tipo “boolean” y esto será igual a la variable “auxNumber” y con un punto (.) llamamos la función some(), entre los paréntesis creamos un parámetro para que nos recorra el array, “num”, y le decimos que si “num” sea mayor a “7”. En código nos queda de la siguiente manera =>

```
const someAuxNumber: boolean = auxNumber.some((num) => num > 7);
```

Para mostrar el mensaje usamos “console.log()”, llamamos la constante “someAuxNumber” y le damos un alias “some” =>

```
console.log(someAuxNumber, 'some');
```

El resultado de esto sería lo siguiente =>

```
true some
```

Esto nos quiere decir que en el arreglo hay al menos un valor mayor a 7

EJERCICIO 2°

Para este ejercicio usaremos el siguiente arreglo tipo “String”, usamos console.log para mostrar el array y le damos un alias denominado “string” =>

```
const auxString: string[] = ['1', '2', '3', '4', '5', '6', '7'];
console.log(auxString, 'string');
```

En consola se vería de la siguiente manera =>

```
[
  '1', '2', '3',
  '4', '5', '6',
  '7'
] string
```

1) agregar al array por medio de push los siguientes valores 10, 11, 12, 13, 14 de tipo string

Para saber que un elemento es de tipo “String” debemos encerrarlo entre comillas simples (‘ ’). Para este ejercicio usaremos la función push de la siguiente manera, primero llamamos la variable del arreglo “auxString” y mediante un punto (.) asignamos la función push(), entre los paréntesis van los valores que deseamos agregar, estos para ser tipo string deben ir obligatoriamente entre comillas simples, quedando de la siguiente manera =>

```
auxString.push('10');
auxString.push('11');
auxString.push('12');
auxString.push('13');
auxString.push('14');
```

Para mostrar el resultado usaremos “console.log()”, dónde llamamos el arreglo y le damos un alias en este caso “push” =>

```
console.log(auxNumber, 'push');
```

Dándonos cómo respuesta lo siguiente =>

```
[
  '1',  '2',  '3',  '4',
  '5',  '6',  '7',  '10',
  '11', '12', '13', '14'
] push
```

Aquí podemos ver cómo se guardaron perfectamente los nuevos elementos

2) Encontrar el índice de los elementos 10, 11, 14 con la función indexOf()

Cómo mencionamos anteriormente para hacer este ejercicio primero creamos nuevas constantes donde se guarden los índices en este caso sería los siguientes: }2indexAuxString10, indexAuxString11, indexAuxString14" luego de esto empleamos la función de tal manera, llamamos al arreglo y mediante un punto (.) usamos la función indexOf() y entre los paréntesis ponemos los elementos entre comillas simples. Quedando de la siguiente manera =>

```
const indexAuxString10 = auxString.indexOf('10');
const indexAuxString11 = auxString.indexOf('11');
const indexAuxString14 = auxString.indexOf('14');
```

Mediante un console.log imprimimos el resultado, le asignaremos un alias "indexOf" =>

```
console.log(indexAuxString10, indexAuxString11, indexAuxString14, 'indexOf');
```

Esto impreso debe dar lo siguiente =>

```
7 8 11 indexOf
```

Esto quiere decir que en las posiciones 7, 8, 11 están nuestros elementos respectivamente

EJERCICIO 3°

Para este ejercicio utilizaremos una cadena de texto e imprimimos con console.log() =>

```
const message: string = 'Bienvenido al Itp';
console.log(message, 'string');
```

1) Dividir la cadena en un array de palabras donde el separador sea un espacio mediante la función split()

La función Split() nos permite convertir una cadena de texto en un arreglo, para este ejercicio primero creamos una constante dónde se guarden los datos, la he denominado "splitMessage" a este le decimos que va a ser un "array" de "string" o un arreglo de una cadena de texto, luego llamamos la variable dónde se encuentra nuestro "string" y mediante un punto (.) podemos emplear la función Split(), entre los paréntesis y por medio de comillas simples (' ') le decimos qué nuestro separador va a ser un espacio. En código se vería así =>

```
const splitMessage: string[] = message.split(' ');
```

Mediante console.log() imprimiremos el resultado obtenido y le damos una alias 'split' =>

```
console.log(splitMessage, 'split');
```

Esto nos debe retornar lo siguiente =>

```
[ 'Bienvenido', 'al', 'Itp' ] split
```

Cómo podemos ver, la función Split() a separado cada palabra en la cuál cada espacio a sido eliminado y la ha insertado en un array separado por comas, de tal manera que cada una de ellas sea independiente de la otra.

2) A ese resultado aplicar la función join para separarlos por comas (,)

Ahora convertiremos el arreglo “splitMessage” en un string, pero esta vez no será separado por espacios sino por comas (,). Para esto creamos una constante al que la llamaremos “joinedSplitMessage” al que la asignamos de tipo string, esta constante será igual a la variable “splitMessage” y mediante un punto (.) usamos la función join() y entre los paréntesis le decimos con que queremos que sean unidos, en este caso será por comas (,). Quedando nuestro código de la siguiente manera =>

```
const joinedSplitMessage: string = splitMessage.join(',');
```

Mediante console.log() podremos imprimir nuestro resultado, el alias que le añadiremos será ‘join’ =>

```
console.log(joinedSplitMessage, 'join');
```

Dando como resultado lo siguiente =>

```
Bienvenido,al,Itp join
```

Cómo podemos ver tenemos las mismas palabras, pero esta vez cada una es separada por una coma (,)

EJERCICIO 4°

Para este ejercicio usaremos el siguiente código =>

```
const names: string = 'Michael, Anderson, Yadir, Kevin, Emerson';
console.log(names, 'string');
```

1) Dividir la cadena de texto en un array de palabras donde el separador sean las comas(,) con la función Split()

Para este ejercicio asignamos una constante llamada “splitNames” al que le damos un tipado de un arreglo de string[], esa constante será igual a la variable “names” y con un punto podemos agregarle la función Split(), entre paréntesis va lo que vamos a usar como nuestro separador, en este caso la coma(,). Mediante console.log() imprimiremos nuestro resultado y la damos un alias “Split” =>

```
const splitNames: string[] = names.split(',');
console.log(splitNames, 'split');
```

Esto nos da cómo resultado lo siguiente =>

```
[ 'Michael', ' Anderson', ' Yadir', ' Kevin', ' Emerson' ] split
```

Cómo podemos ver al arreglo le a añadido cada valor por el que en su momento fue separado por una coma (,)

2) Aplicar la función join() pero este debe ser separado por espacios

Para este ejercicio primero creamos una constante “joinedSplitNames” de tipo “string”, luego hacemos llamado de la constante anterior “splitNames” y con un punto (.) agregamos la función join(), entre sus paréntesis va lo que emplearemos como nuestro separador. Y mediante console.log() imprimiremos nuestro resultado, a este le asignamos un alias ‘join’ =>

```
const joinedSplitNames: string = splitNames.join(' ');
console.log(joinedSplitNames, 'join');
```

Dándonos como resultado lo siguiente =>

```
Michael  Anderson  Yadir  Kevin  Emerson join
```

Cómo podemos ver cada palabra a sido unido por la función join, pero separadas por un espacio entre ellas.

EJERCICIO 5°

Para este ejercicio usaremos el siguiente objeto =>

```
const arrayProducts: any[] = [{
  id: 1,
  name: 'arroz',
  price: 1000,
  priceDiscount: 700,
},
{
  id: 2,
  name: 'atún',
  price: 2500,
  priceDiscount: 1500,
}];
console.log(arrayProducts);
```

1) filtrar en una constante donde price sea mayor de 900 por medio de la función filter() y el ciclo for()

a) Filter()

Para este parte del ejercicio usaremos la función filter() ya que bien cómo dice su nombre es un filtro, este filtro lo emplearemos para poder obtener los elementos que queremos. Para ello primero creamos una constante dónde se guardarán los elementos, a este lo hemos denominado "filterNumber" para lograr el resultado entonces hacemos los siguiente llamamos al objeto y mediante un punto (.) añadiremos la función filter(), luego creamos un parámetro el cuál recorrerá cada elemento de nuestro objeto, a este le decimos que en el atributo price que sea mayor a 900 lo guarde. Para ver el resultado usamos console.log() y le damos un alias 'filter' =>

```
const filterNumber = arrayProducts.filter((num) => num.price > 900);
console.log(filterNumber, 'filter');
```

Dándonos cómo resultado lo siguiente =>

```
[
  { id: 1, name: 'arroz', price: 1000, priceDiscount: 700 },
  { id: 2, name: 'atún', price: 2500, priceDiscount: 1500 }
] filter
```

Aquí podemos ver qué ha traído los elementos del objeto los cuáles sean mayores 900

b) Ciclo for

Para resolver esto de esta manera lo que debemos hacer es crear una constante que nos permita guarda aquellos elementos que estemos buscando, para esta he decidido nombrarla "forArraysProducts" y creamos una arreglo vacío de tipo cualquiera (any), luego hacemos un ciclo for() común y corriente dónde declaramos cómo let un auxiliar "i" que inicie en 0 pero que recorrerá hasta un antes del total del tamaño del arreglo principal en este caso "arrayProducts", y que irá de 1 en 1, luego de hacer esto, empleamos un if() este nos permite hacer una condición en la cuál si se cumple se hará lo que pedimos, de lo contrario no, en esta condición lo que haremos es que el atributo price del arreglo "arrayProducts" en la posición de "i" sea mayor a 900, si se cumple entonces emplearemos la función push() que guardará el elemento en nuestro arreglo vacío. Y luego imprimiremos esto mediante un console.log con un alias 'cycle for'. Después de hacer esto nuestro código debe quedar de la siguiente manera =>

```
const forArraysProducts : any[] = [];
for (let i = 0; i < arrayProducts.length; i += 1) {
  if (arrayProducts[i].price > 900) {
    forArraysProducts.push(arrayProducts[i]);
  }
}
console.log(forArraysProducts, ' cycle for');
```

El resultado que nos mostrará es el siguiente =>

```
[
  { id: 1, name: 'arroz', price: 1000, priceDiscount: 700 },
  { id: 2, name: 'atún', price: 2500, priceDiscount: 1500 }
]  cycle for
```

2) Encontrar el primer objeto del array donde el priceDiscount sea mayor a 500 mediante la función find()

Cómo bien se traduce find() nos permite encontrar un valor dentro de un array mediante una petición. Para resolver este ejercicio empezaremos como anteriormente hemos hecho, declarando una constante que nos permita guardar el resultado final, este la he denominado cómo "foundArrayProducts" este lo definiremos cómo un tipo "number" (número) e "undefined" (indefinido), luego llamamos a nuestro arreglo y mediante un punto (.) emplearemos la función find(), entre sus paréntesis creamos un parámetro para que nos permita recorrer el arreglo, a este le decimos que seleccione el primer atributo "priceDiscount" que sea mayor a 500. Esto será mostrado a través de un console.log() con un alias denominado 'find'. Nuestro código debe de ser así =>

```
const foundArrayProducts: number | undefined =
arrayProducts.find((num) => num.priceDiscount >
500);
console.log(foundArrayProducts, 'find');
```

Esto es lo que nos debe retornar lo siguiente en nuestra consola =>

```
{ id: 1, name: 'arroz', price: 1000, priceDiscount: 700 } find
```

Aquí podemos observar que el primer elemento el cuál "priceDiscount" sea mayor a 500.

3) Encontrar el índice del primer elemento del array donde el price sea mayor a 1000 con la función findindex()

Este es similar al anterior, pero en vez de traer el elemento, trae la posición del elemento. Lo único que cambia en su estructura es la función que emplearemos en este caso findIndex(), declaramos una constante "foundIndexArrayProducts" y un parámetro el cual buscara el primer atributo Price el cual sea mayor a 1000. Imprimimos en console.log y le damos un alias 'findIndex' para este caso =>

```
const foundIndexArrayProducts: number =
arrayProducts.findIndex((num) => num.price > 1000);
console.log(foundIndexArrayProducts, 'findIndex');
```

Nuestro resultado será el siguiente =>

```
1 findIndex
```

Cómo podemos ver la posición del primer elemento en el cual el atributo “Price” sea mayor a 1000, es la posición 1.

4) Agregar 3 objetos nuevos de un producto de una tienda por medio de la función push()

Para agregar los nuevos productos por medio de la función push() debemos llamar al arreglo principal y mediante un punto podemos usar la función push(), entre sus paréntesis agregamos el producto siguiendo su estructura inicial. Luego miramos el resultado mediante un console.log() y su alias será ‘push’ =>

```
arrayProducts.push(
  {
    id: 3, name: 'mayonesa', price: 5000, priceDiscount: 2000,
  },
);
arrayProducts.push(
  {
    id: 4, name: 'ketchup', price: 4500, priceDiscount: 1800,
  },
);
arrayProducts.push(
  {
    id: 5, name: 'mostaza', price: 3500, priceDiscount: 1500,
  },
);
console.log(arrayProducts, 'push');
```

Nuestro resultado en consola será el siguiente =>

```
[
  { id: 1, name: 'arroz', price: 1000, priceDiscount: 700 },
  { id: 2, name: 'atún', price: 2500, priceDiscount: 1500 },
  { id: 3, name: 'mayonesa', price: 5000, priceDiscount: 2000 },
  { id: 4, name: 'ketchup', price: 4500, priceDiscount: 1800 },
  { id: 5, name: 'mostaza', price: 3500, priceDiscount: 1500 }
] push
```

Aquí vemos cómo se han agregado los nuevos elementos a nuestro objeto.

EJERCICIO 6°

Para este ejercicio emplearemos 2 arreglos, los cuales serán =>

```
const array01: number[] = [1, 2, 3, 4];
const array02: number[] = [6, 7, 8, 9];
```

1) concatenar el array01 y array02 con las funciones concat() y push()
a) Concat()

Para este ejercicio lo que haremos es crear una constante para agregar nuestro resultado final, en este caso la llamaremos “concatArray03” de tipo “number” pero que sea un arreglo. Luego de crear esa constante llamamos nuestro primer arreglo “array01” y mediante un punto (.) usamos la función concat(), entre sus paréntesis agregaremos el segundo arreglo que queramos unir. Mediante un console.log() mostramos el resultado y le asignamos un alias ‘concat’ =>

```
const concatArray03 : number[] = array01.concat
(array02);
console.log(concatArray03, 'concat');
```

El resultado final deberá ser el siguiente =>

```
[
  1, 2, 3, 4,
  6, 7, 8, 9
] concat
```

Cómo podemos ver se han unido o concatenado los dos arreglos en uno sólo.

b) Push()

Para usar la función push() en este ejercicio lo que haremos es, crear una constante qué nos guardará la cantidad de datos que han sido agregados, la he llamado “ pushArray03” esto debe ser igual al “array01”, con un punto (.) agregamos la función push y entre sus paréntesis llamamos al “array02” con tres puntos adelante, estos tres puntos significa que se agregaran varios elementos al arreglo “array01”. Ahora emplearemos dos console.log, en uno llamaremos la constante creada con su alias respectivo ‘push’ y en el segundo llamamos el arreglo al cual se le insertaron los elementos del otro arreglo y también con su respectivo alias ‘new array01 value’. Quedándonos de la siguiente manera =>

```
const pushArray03 = (array01.push(...array02));
console.log(pushArray03, 'push');
console.log(array01, 'new array01 value');
```

Cómo resultado debe darnos esto =>

```
8 push
[
  1, 2, 3, 4,
  6, 7, 8, 9
] new array01 value
```

Aquí vemos que en push nos sale la cantidad de elementos que hay en el arreglo, y en el siguiente vemos el arreglo con los valores insertados.

2) A la respuesta de concatArray03 la ordenaremos de menor a mayor mediante la función sort()

Este ejercicio se manejará como hemos venido resolviendo los anteriores, en la que debemos crear una constante tipo arreglo de números para guardar el resultado final, la he nombrado de la siguiente manera “sortedArrays”, esto debe ser igual a la respuesta antes generada mediante “concatArray03” y con un punto (.) agregamos la función sort(), dónde en sus paréntesis haremos dos parámetros uno “a” y otro “b” al que le decimos que “a” debe ser menor que “b” y se irá guardando de posición en posición hasta tener el resultado que queremos. También emplearemos un console.log para imprimir y le asignamos un alias ‘sort<>’ =>

```
const sortedArrays: number[] = concatArray03.sort
((a, b) => a - b);
console.log(sortedArrays, 'sort<>');
```

Dándonos cómo resultado lo siguiente =>

```
[
  1, 2, 3, 4,
  6, 7, 8, 9
] sort<>
```

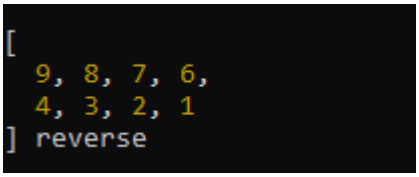
Aquí vemos que los valores numéricos han sido ordenados de menor a mayor.

3) ordenar de forma descendente por medio de la función reverse()

Para resolver este ejercicio crearemos una constante de tipo arreglo numérico “reversedArrays”, entonces para conseguir el resultado que queremos utilizamos la respuesta de “sortedArrays” y mediante un punto(.) agregamos la función reverse(), esta función lo que hace es qué ubicará cada elemento teniendo en cuenta su valor, de forma que el numero mayor irá en la posición 0 y el número menor irá en la última posición. Mediante un console.log revisaremos el resultado y le damos su alias respectivo ‘reverse’ =>

```
const reversedArrays: number[] = sortedArrays.
reverse();
console.log(reversedArrays, 'reverse');
```

Nuestro resultado debe ser el siguiente =>



Aquí podemos ver qué los números de nuestro arreglo ha sido ordenado de mayor a menor.

GitHub:

https://github.com/EstP19/Programaci-n_backend/tree/main/clase_ts/documentaci%C3%B3n