Técnicas de Programación

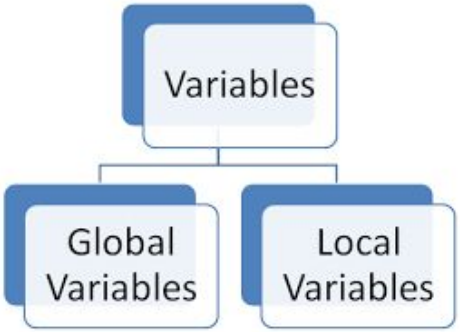
Programador full-stack

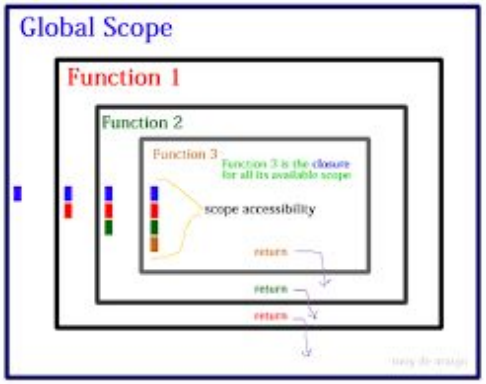
Ámbito Variable, Legibilidad, Arreglos (Repaso)

Ámbito de las Variables

Al utilizar funciones se establece un límite para el alcance de las variables

- **Variables Locales**: Son aquellas que se encuentran dentro de un método. El valor se confina al método en el que está declarada
- **Variables Globales**: Son las que se definen o están declaradas en el algoritmo principal. Pueden utilizarse en cualquier método
- Se debe intentar crear métodos con variables locales y pocos parámetros para favorecer la reutilización y el mantenimiento del software





Buenas Prácticas de Programación

- Entender el problema, diseñar una estrategia, implementar
- Nombres representativos de variables y métodos
- Código claro, comprensible, etc.
- Identación en las estructuras de control
- Comentarios en el código
- *//Así se comenta en TypeScript, con las dos barras*

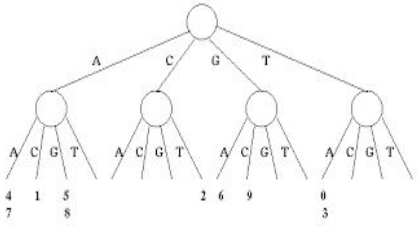
Buenas Prácticas de Programación

- Usar métodos
- No duplicar código
- Dividir el problema en sub problemas
- Construir el código tan simple como sea posible
- Que el código funcione no significa que esté bien programado

Estructuras de Datos

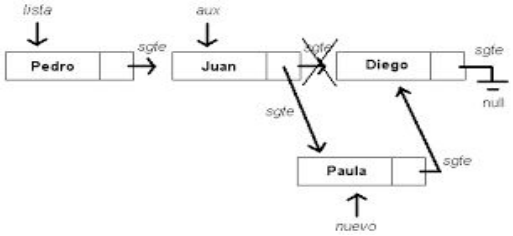
Forma particular de organizar datos

- Estructuras que permiten **COLECCIONAR** elementos
  - GUARDARLOS
  - RECORRERLOS
  - MANIPULARLOS



- Operaciones básicas
  - **COLOCAR**
  - **OBTENER**

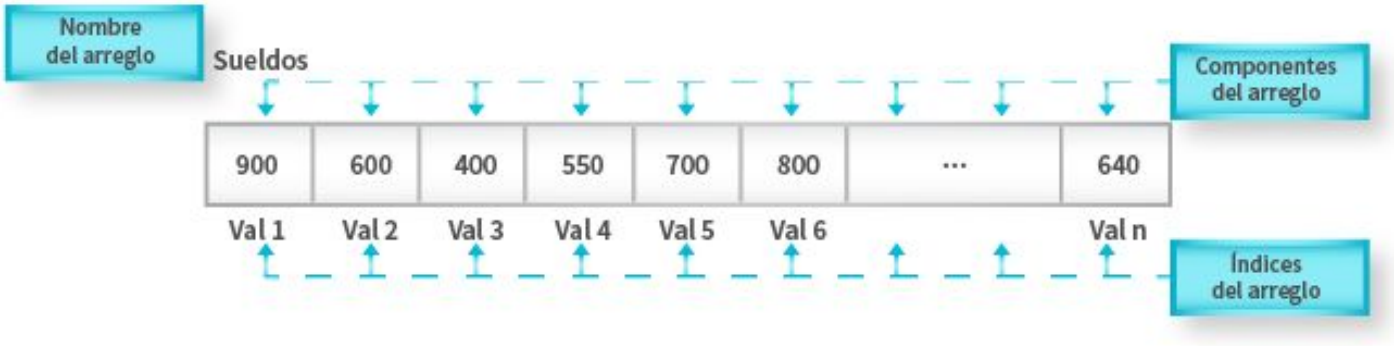
- Estructuras
  - **LISTAS**
  - **COLAS**
  - **PILAS**
  - **ARBOLES**



Estructuras de Datos

Arreglos/Listas/Vectores

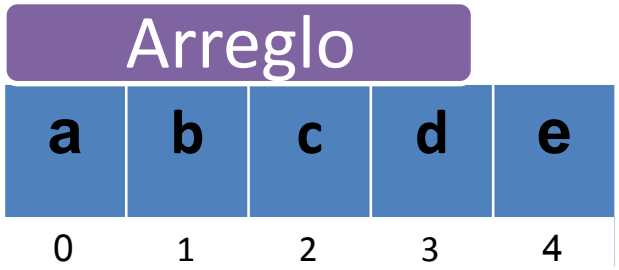
- Los arreglos son estructuras de datos homogéneas (todos sus datos son del mismo tipo)
- Permiten almacenar un determinado número de datos
- Tiene muchos elementos, y a cada uno de ellos se acceden indicando que posición se quiere usar (un índice)



Estructuras de Datos

Arreglos/Listas/Vectores

- Lista = Array = Vector
- Todos los elementos deben ser del mismo tipo de dato
- TS: Zero-based (arreglos de base cero) -> Índices comienzan en 0
- Length indica la cantidad total de elementos (también es igual a la posición del último elemento más 1)
- Propiedades de los arreglos:
  - ELEMENTO o ITEM: a, b, c, d, e
  - LONGITUD: 5
  - INDICE o SUBINDICE: 0, 1, 2, 3, 4



Longitud = Length= 5

Técnicas de Programación

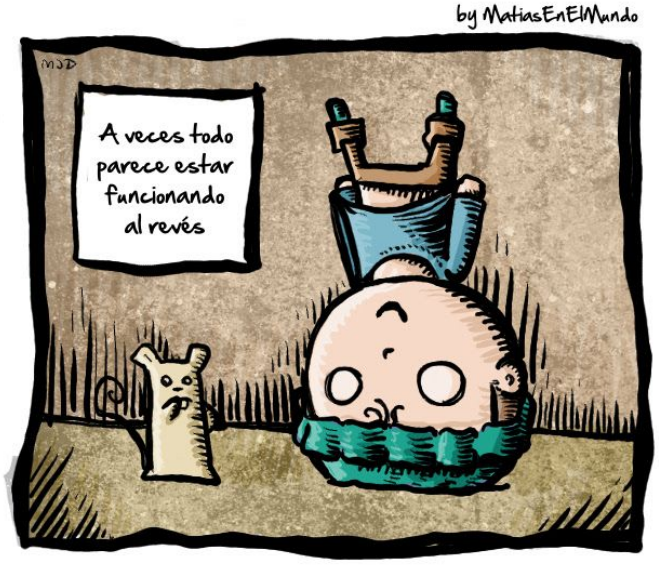
Programador full-stack

Arreglos (Ejercicios)

Estructuras de Datos

Invertir Arreglo

- **Almacene** en un arreglo de tamaño **N** los números ingresados por el usuario
- La **dimensión N** también es ingresada por el usuario
- **Muestre** los números del arreglo pero del último al primero



Ejemplo:

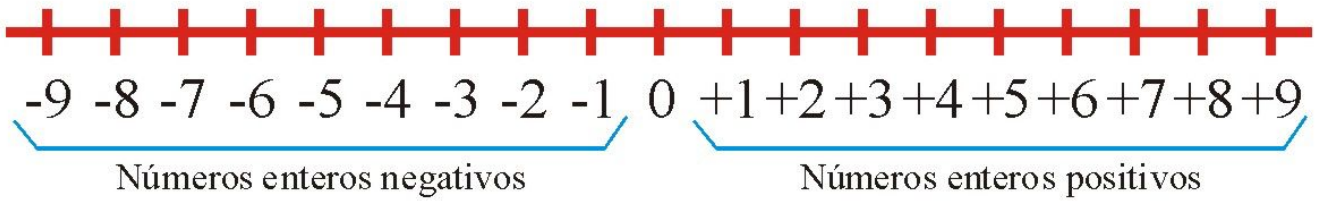
v = 1, 3, 7, 9, 9, 5
La salida es: 5, 9, 9, 7, 3, 1

Estructuras de Datos

Tipos de Números en Arreglo

- Almacene en un arreglo de dimensión N números (la cantidad es ingresada por el usuario)
- Muestre cuántos números son positivos, cuántos son negativos y cuántos ceros hay

Ejemplo:
v = 0, -7, -9, 1, 0, 0
La salida es: 1 positivos, 2 negativos y 3 ceros



Técnicas de Programación

Programador full-stack

Arreglos (Resolución)

Estructuras de Datos

Sumar Arreglos

```
//Algoritmo SumarArreglos
import * as rls from 'readline-sync';

let v1 : number[] = new Array(6);
let v2 : number[] = new Array(6);
let vSuma : number[] = new Array(6);
let indice : number;
//Cargo el vector v1
for (indice = 0; indice < 6; indice++) {
        v1[indice]=rls.questionInt(`Ingrese el valor de v1[ ${indice} ]`);
}
//Cargo el vector v2
for (indice = 0; indice < 6; indice++) {
        v2[indice]=rls.questionInt(`Ingrese el valor de v2[ ${indice} ]`);
}
//Sumo los valores y muestro
for (indice = 0; indice < 6; indice++) {
        vSuma[indice] = v1[indice] + v2[indice];
        console.log (`vSuma[ ${indice} ]= ${vSuma[indice]}`);
}
```

$$A + B = \langle (a_1 + a_2), (b_1 + b_2), (c_1 + c_2) \rangle$$

ex.

$A = \langle 5, 9, -10 \rangle \quad B = \langle 17, -3, -2 \rangle$

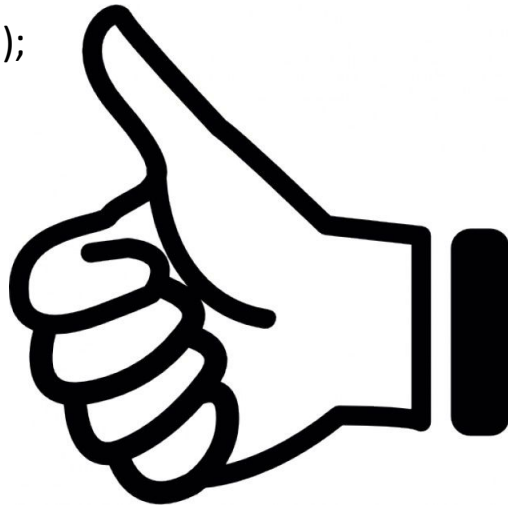
$A+B = \langle (5+17), (9+(-3)), ((-10)+(-2)) \rangle$

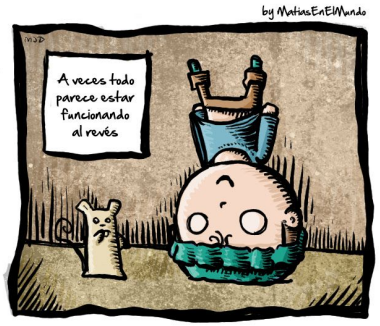
$= \langle 22, 6, -12 \rangle$

Estructuras de Datos

Invertir Arreglo

```
//Algoritmo InvertirArreglo
import * as rls from 'readline-sync';

let cantidad : number = rls.questionInt(`Ingrese la cantidad de números:`);
let v : number[] = new Array (cantidad);
let indice : number;
for (indice = 0; indice<cantidad; indice++) {
        v[indice] = rls.questionInt(`Ingrese v[ ${indice} ]: `));
}
let cadena : string ="";
for (indice = cantidad-1; indice>=0; indice--) {
        cadena = cadena + v[indice] + " ";
}
console.log(cadena);
```





Estructuras de Datos

Tipos de Números en Arreglo

```
//Algoritmo TiposNumero
import * as rls from 'readline-sync';

let cantidad : number = rls.questionInt(`Ingrese la cantidad de números: `);
let v : number[] = new Array (cantidad);
let indice : number;
for (indice = 0; indice < cantidad; indice++) {
        v[indice] = rls.questionInt(`Ingrese v[ ${indice} ]: `));
}
let numNeg : number = 0;
let numCero : number = 0;
let numPos : number = 0;
for (indice = 0; indice < cantidad; indice++) {
        if (v[indice] == 0) {
                numCero++;
        } else if (v[indice] > 0) {
                numPos++;
        } else {
                numNeg++;
        }
}
console.log (`Hay ${numPos} positivos, ${numNeg} negativos, ${numCero} ceros.`);
```

Técnicas de Programación

Programador full-stack

Estructuras de Datos y Métodos (Conceptos)

Estructuras de Datos

Sumar Arreglos

- Sumar los elementos de cada una de las posiciones de dos vectores y guardar el resultado en otro vector
- El arreglo tiene dimensión 6 y los números de los dos vectores los carga el usuario

Ejemplo:

v1 = 1, 3, 7, 9, 9, 5
v2 = 6, 9, 2, 5, 9, 4
vSuma = 7, 12, 9, 14, 18, 9

$$A + B = \langle (a_1 + a_2), (b_1 + b_2), (c_1 + c_2) \rangle$$

ex.

$A = \langle 5, 9, -10 \rangle \quad B = \langle 17, -3, -2 \rangle$

$A+B = \langle (5+17), (9+(-3)), ((-10)+(-2)) \rangle$

$= \langle 22, 6, -12 \rangle$

Estructuras de Datos

Sumar Arreglos

1) Definir las variables:

```
let v1 : number[] = new Array(6);

let v2 : number[] = new Array(6);

let vSuma : number[] = new Array(6);

let indice : number;
```

$$A + B = \langle (a_1 + a_2),(b_1 + b_2),(c_1 + c_2) \rangle$$

ex.

$A = \langle 5, 9, -10 \rangle \quad B = \langle 17, -3, -2 \rangle$

$A+B = \langle (5+17),(9+(-3)),((-10)+(-2)) \rangle$

$= \langle 22, 6, -12 \rangle$

Estructuras de Datos

Sumar Arreglos

2) Cargar los valores:

```
//Cargo el vector v1
for (indice=0; indice<6; indice++) {

        v1[indice] = readline.questionInt("Ingrese el valor de v1[", indice, "]: ");

}

//Cargo el vector v2
for (indice=0; indice<6; indice++) {

        v2[indice] = readline.questionInt("Ingrese el valor de v2[", indice, "]: ");

}
```

$$A + B = \langle (a_1 + a_2),(b_1 + b_2),(c_1 + c_2) \rangle$$

ex.

$A = \langle 5, 9, -10 \rangle \quad B = \langle 17, -3, -2 \rangle$

$A+B = \langle (5+17),(9+(-3)),((-10)+(-2)) \rangle$

$= \langle 22, 6, -12 \rangle$

Estructuras de Datos

Sumar Arreglos

3) Sumar los valores:

```
//Sumo los valores y muestro
for (indice=0; indice<6; indice++) {

        vSuma[indice] = v1[indice] + v2[indice];

        console.log("vSuma[", indice, "]=", vSuma[indice]);

}
```

$$A + B = \langle (a_1 + a_2),(b_1 + b_2),(c_1 + c_2) \rangle$$

ex.

$A = \langle 5, 9, -10 \rangle \quad B = \langle 17, -3, -2 \rangle$

$A+B = \langle (5+17),(9+(-3)),((-10)+(-2)) \rangle$

$= \langle 22, 6, -12 \rangle$

Estructuras de Datos

Sumar Arreglos

```
import * as rls from 'readline-sync';

//Declaro las variables
let v1 : number[] = new Array(6);
let v2 : number[] = new Array(6);
let vSuma : number[] = new Array(6);
let indice : number;

//Cargo el vector v1
for (indice=0; indice<6; indice++) {
        v1[indice] = rls.questionInt("Ingrese el valor de v1[", indice, "]: ");
}
//Cargo el vector v2
for (indice=0; indice<6; indice++) {
        v2[indice] = rls.questionInt("Ingrese el valor de v2[", indice, "]: ");
}
//Sumo los valores y muestro
for (indice=0; indice<6; indice++) {
        vSuma[indice] = v1[indice] + v2[indice];
        console.log("vSuma[", indice, "]=", vSuma[indice]);
}
```

$$A + B = \langle (a_1 + a_2), (b_1 + b_2), (c_1 + c_2) \rangle$$

ex.

$A = \langle 5, 9, -10 \rangle \quad B = \langle 17, -3, -2 \rangle$

$A+B = \langle (5+17), (9+(-3)), ((-10)+(-2)) \rangle$

$= \langle 22, 6, -12 \rangle$

Estructuras de Datos

Invertir Arreglo

- **Almacene** en un arreglo de tamaño **N** los números ingresados por el usuario
- La **dimensión N** también es ingresada por el usuario
- **Muestre** los números del arreglo pero del último al primero

Ejemplo:

v = 1, 3, 7, 9, 9, 5
La salida es: 5, 9, 9, 7, 3, 1

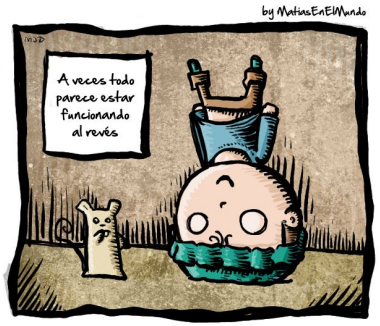
Estructuras de Datos

Invertir Arreglo

1) Definir las variables:

```
let cantidad : number = rls.questionInt("Ingrese la cantidad de números:");

let v : number[] = new Array (cantidad);

let indice : number;
```

El tamaño del arreglo no es fijo, depende de la entrada del usuario (o el valor de una variable)



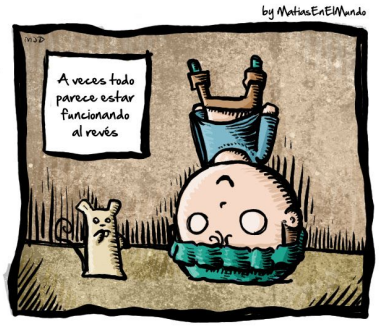
Estructuras de Datos

Invertir Arreglo

2) Cargar el arreglo:

```
for (indice = 0;  indice < cantidad  ; indice++) {

        v[indice] = rls.questionInt("Ingrese v[", indice, "]");

}
```

Como el tamaño del arreglo es desconocido, utilizamos *cantidad* como límite de la instrucción **for**



Estructuras de Datos

Invertir Arreglo

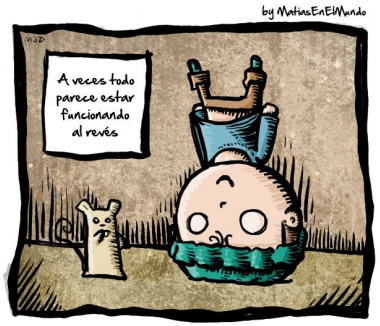
3) Mostrar al revés:

Empiezo desde el último elemento

Y me detengo en el primero

Restando de a uno

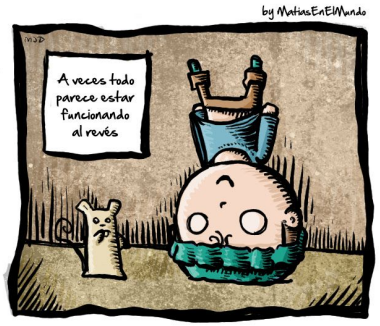
```
for (  indice = cantidad - 1;      indice >= 0;      indice--      ) {

      console.log(v[indice], " ");

}
```



Estructuras de Datos

Invertir Arreglo

```
import * as rls from 'readline-sync';

let cantidad : number = rls.questionInt("Ingrese la cantidad de números:");

let v : number[] = new Array (cantidad);

let indice : number;

for (indice = 0;  indice < cantidad ; indice++) {
        v[indice] = rls.questionInt("Ingrese v[", indice, "]");
}

for (indice = cantidad - 1; indice >= 0; indice-- ) {
        console.log(v[indice], " ");
}
```



Estructuras de Datos

Errores típicos en manejo de Arreglos

```
let indice : number = 0;
while (indice < tamañoarreglo) {
        arreglo[indice] = rls.questionInt("Ingrese número arreglo[", indice, "]:");
}
```

No se incrementa el contador usado para controlar el ciclo, por lo tanto produce un ciclo infinito.

```
let indice : number;
for (indice = 0;  indice<tamañoarregloIMPAR ; indice+=2) {
        arreglo[indice] = rls.questionInt("Ingrese número arreglo[", indice, "]:");
}
```

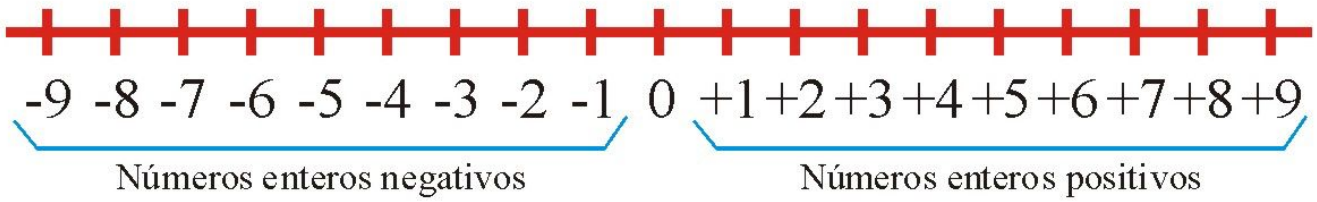
El incremento del contador usado para controlar el ciclo hace que el valor no coincida con el límite impuesto lo que puede producir un comportamiento inesperado.

Estructuras de Datos

Tipos de Números en Arreglo

- Almacene en un arreglo de dimensión N números (la cantidad es ingresada por el usuario)
- Muestre cuántos números son positivos, cuántos son negativos y cuántos ceros hay

Ejemplo:
v = 0, -7, -9, 1, 0, 0
La salida es: 1 positivos, 2 negativos y 3 ceros



Estructuras de Datos

Tipos de Números en Arreglo

```
let i : number, positivos : number, negativos : number, ceros : number;
let v : number[] = new Array(dimension_arreglo);

for ( i=0 ; i < dimension_arreglo; i++) {
        arreglo[i]= rls.questionInt("ingrese un numero");

        if (arreglo[i] > 0) {
                positivos++;
        } else if (arreglo[i] < 0) {
                negativos++;
        } else {
                ceros++;
        }
}
```

Dentro del **ciclo** se hacen dos cosas diferentes!

1) **Cargar el arreglo**
2) **Contar los tipos de enteros**

Esto afecta la modularidad del código y limita su refactorización

Estructuras de Datos

Tipos de Números en Arreglo

1) Defino las variables y cargo los números en el arreglo

```
let cantidad : number = rls.questionInt("Ingrese la cantidad de números:");

let indice : number;

let v : number[] = new Array(cantidad);

for ( indice=0; indice < cantidad; indice++) {

        v[indice] = rls.questionInt("Ingrese v[", indice, "]");

}
```

Estructuras de Datos

Tipos de Números en Arreglo

2) Defino las variables para llevar la cuenta:

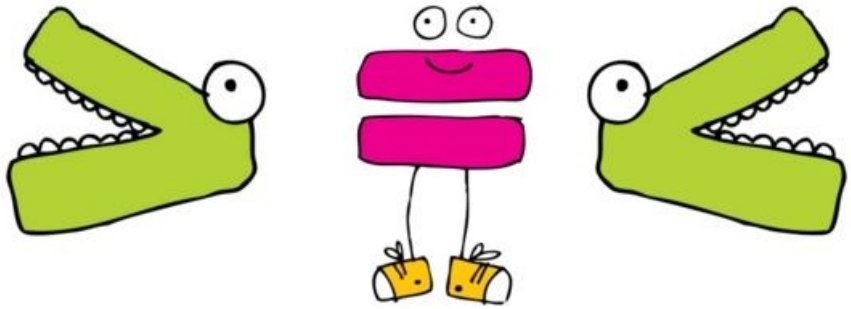
```
//Cuento los valores >0, <0 e =0
let numNeg : number = 0;
let numCero : number = 0;
let numPos : number = 0;
```

Estructuras de Datos

Tipos de Números en Arreglo

3) Recorro el arreglo y voy contando según corresponda

```
for (indice = 0 ; indice < cantidad; indice++) {
        if (v[indice] > 0) {
                numPos++;
        } else if (arreglo[indice] < 0) {
                numNeg++;
        } else {
                numCero++;
        }
}
```



Estructuras de Datos

Tipos de Números en Arreglo

```
import * as rls from 'readline-sync';

let cantidad : number;

cantidad = rls.questionInt("Ingrese la cantidad de números:");

let indice : number;
let v : number[] = new Array(cantidad);

for ( indice=0; indice < cantidad; indice++) {
        v[indice]= rls.questionInt("Ingrese v[", indice, "]"));
}
//Cuento los valores >0, <0 e =0
let numNeg : number = 0;
let numCero : number = 0;
let numPos : number = 0;

// sigue
```

```
for (indice = 0 ; indice < cantidad; indice++) {
        if (v[indice] > 0) {
                numPos++;
        } else if (arreglo[indice] < 0) {
                numNeg++;
        } else {
                numCero++;
        }
}
console.log(numPos, " positivos, ");
console.log(numNeg, " negativos, ");
console.log(numCero, " ceros");
```

Estructuras de Datos

Arreglos, Métodos y Pasaje de Parámetros

- Podemos **reutilizar** código!
- Las **modificaciones** se pueden hacer **directamente** en los arreglos que pasamos como **parámetro** (solo funciona para arreglos y matrices, no para otros tipos de datos)

Estructuras de Datos

Pasos para Migrar a Métodos

1. Identificar código repetido o funcionalidad “reusable”
2. Identificar parámetros comunes y retorno (si fuese necesario devolver un resultado)
3. Modificar el código para aprovechar el código mejorado (por ejemplo, la carga de un vector o la escritura por pantalla)

Estructuras de Datos

Sumar Arreglos (con Métodos)

- Sumar los elementos de cada una de las posiciones de dos vectores y guardar el resultado en otro vector
- El arreglo tiene dimensión 6 y los números de los dos vectores los carga el usuario

Ejemplo:

v1 = 1, 3, 7, 9, 9, 5
v2 = 6, 9, 2, 5, 9, 4
vSuma = 7, 12, 9, 14, 18, 9

$$A + B = \langle (a_1 + a_2), (b_1 + b_2), (c_1 + c_2) \rangle$$

ex.

$A = \langle 5, 9, -10 \rangle \quad B = \langle 17, -3, -2 \rangle$

$A+B = \langle (5+17), (9+(-3)), ((-10)+(-2)) \rangle$
$= \langle 22, 6, -12 \rangle$

Estructuras de Datos

Sumar Arreglos (con Métodos)

```
let v1 : number[] = new Array(6);
let v2 : number[] = new Array(6);
let vSuma : number[] = new Array(6);
let indice : number;

//Cargo el vector v1
for (indice=0; indice<6; indice++) {
        v1[indice] = rls.questionInt("Ingrese el valor de v1[", indice, "]: ");
}
//Cargo el vector v2
for (indice=0; indice<6; indice++) {
        v2[indice] = rls.questionInt("Ingrese el valor de v2[", indice, "]: ");
}
//Sumo los valores y muestro
for (indice=0; indice<6; indice++) {
        vSuma[indice] = v1[indice] + v2[indice];
        console.log("vSuma[", indice, "]=", vSuma[indice]);
}
```

La carga de los vectores está repetida

La suma puede ser reutilizada (con el **for**)

La escritura por pantalla puede ser reutilizada (con el **for**)

Estructuras de Datos

Sumar Arreglos (con Métodos)

```
function cargarVector(v : number[], cantidad : number) {

        let indice : number;

        for (indice=0; indice<cantidad; indice++) {

                v[indice] = rls.questionInt("Ingrese el valor de la posición ", indice, ": ");

        }
}
```

$$A + B = \langle (a_1 + a_2), (b_1 + b_2), (c_1 + c_2) \rangle$$

ex.

$A = \langle 5, 9, -10 \rangle \quad B = \langle 17, -3, -2 \rangle$

$A+B = \langle (5+17), (9+(-3)), ((-10)+(-2)) \rangle$

$= \langle 22, 6, -12 \rangle$

Estructuras de Datos

Sumar Arreglos (con Métodos)

```
function mostrarVector(v : number[], cantidad : number) {

        let indice : number;

        for (indice=0; indice<cantidad; indice++) {

                console.log(v[indice], " “);

        }
}
```

$$A + B = \langle (a_1 + a_2), (b_1 + b_2), (c_1 + c_2) \rangle$$

ex.

$A = \langle 5, 9, -10 \rangle \quad B = \langle 17, -3, -2 \rangle$

$A+B = \langle (5+17), (9+(-3)), ((-10)+(-2)) \rangle$

$= \langle 22, 6, -12 \rangle$

Estructuras de Datos

Sumar Arreglos (con Métodos)

```
function sumarVector(v1 : number[], v2 : number[], vSuma : number[], cantidad : number) {
        let indice : number;
        for (indice=0; indice<cantidad; indice++) {
                vSuma[indice] = v1[indice] + v2[indice];
        }
}
```

Los cambios en vSuma se hacen en la variable definida externamente, es decir, que cambiamos sus valores en la función para que puedan ser observados en el principal

$$A + B = \langle (a_1 + a_2), (b_1 + b_2), (c_1 + c_2) \rangle$$

ex.

$A = \langle 5, 9, -10 \rangle \quad B = \langle 17, -3, -2 \rangle$

$A+B = \langle (5+17), (9+(-3)), ((-10)+(-2)) \rangle$

$= \langle 22, 6, -12 \rangle$

Estructuras de Datos

Sumar Arreglos (con Métodos)

```
import * as rls from 'readline-sync';

let v1 : number[] = new Array(6);
let v2 : number[] = new Array(6);

let vSuma : number[] = new Array(6);

console.log("Cargando v1");
cargarVector(v1, 6);
console.log("Cargando v2");
cargarVector(v2, 6);

sumarVector(v1, v2, vSuma, 6);
console.log("Valores de v1");
mostrarVector(v1, 6)
console.log("Valores de v2");
mostrarVector(v2, 6)
console.log("Valores de vSuma");
mostrarVector(vSuma, 6)
```

$$A + B = \langle (a_1 + a_2), (b_1 + b_2), (c_1 + c_2) \rangle$$

ex.

$A = \langle 5, 9, -10 \rangle \quad B = \langle 17, -3, -2 \rangle$

$A+B = \langle (5+17), (9+(-3)), ((-10)+(-2)) \rangle$

$= \langle 22, 6, -12 \rangle$

Estructuras de Datos

Invertir Arreglo (con Métodos)

- **Almacene** en un arreglo de tamaño **N** los números ingresados por el usuario
- La **dimensión N** también es ingresada por el usuario
- **Muestre** los números del arreglo pero del último al primero

<u>Ejemplo</u>:

v = 1, 3, 7, 9, 9, 5
La salida es: 5, 9, 9, 7, 3, 1

Estructuras de Datos

Invertir Arreglo (con Métodos)

```
let cantidad : number = rls.questionInt("Ingrese la cantidad de números:");

let indice : number;

let v : number[] = new Array(cantidad);

for (indice = 0;  indice < cantidad ; indice++) {
        v[indice] = rls.questionInt("Ingrese v[", indice, "]");
}

for (indice = cantidad - 1; indice >= 0; indice-- ) {
        console.log(v[indice], " ");
}
```

La carga de los vectores está repetida (y ya la implementamos!)

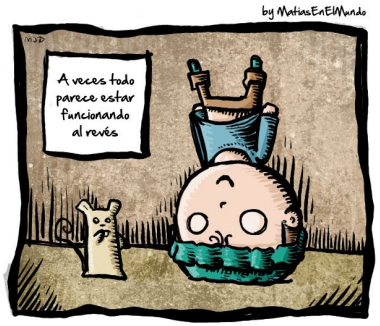
Mostrar invertido podría ser un método nuevo

Invertir un arreglo también podría ser útil (y una alternativa a mostrar invertido)

Estructuras de Datos

Invertir Arreglo (con Métodos)

```
function cargarVector(v : number[], cantidad : number) {

        let indice : number;

        for (indice=0; indice<cantidad; indice++) {

                v[indice] = rls.questionInt("Ingrese el valor en ", indice, ": ");

        }
}
```



Estructuras de Datos

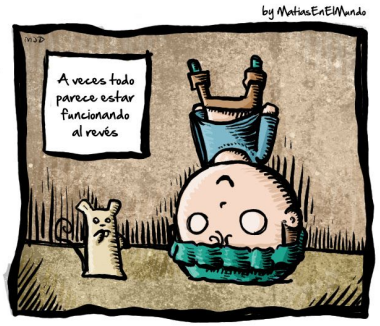
Invertir Arreglo (con Métodos)

```
function mostrarVector(v : number[], cantidad : number) {

        let indice : number;

        for (indice=0; indice<cantidad; indice++) {

                console.log(v[indice], " ");

        }
}
```

Estructuras de Datos

Invertir Arreglo (con Métodos)

```
function mostrarVectorInvertido(v : number[], cantidad : number) {

        let indice : number;

        for (indice=cantidad-1; indice>=0; indice--) {

                console.log(v[indice], " ");

        }
}
```



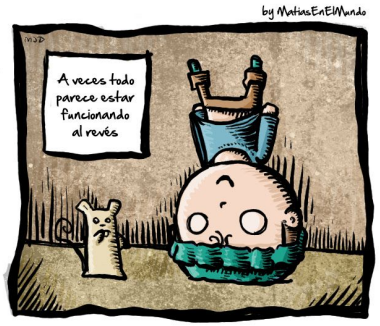
Estructuras de Datos

Invertir Arreglo (con Métodos)

```
function invertirVector(v : number[], cantidad : number) {
        let indiceIzq : number = 0;
        let indiceDer : number = cantidad-1;
        let aux : number;
        while (indiceIzq < indiceDer) {
                aux = v[indiceIzq];
                v[indiceIzq] = v[indiceDer];
                v[indiceDer] = aux;
                indiceIzq++;
                indiceDer--;
        }
}
```

Defino dos índices para moverme desde el primer elemento a la izquierda hacia la derecha y desde el último elemento a la derecha hacia la izquierda

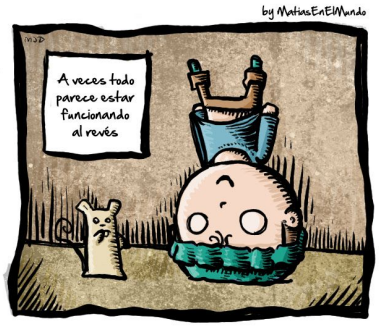
Incremento indiceIzq en 1 y decremento indiceDer en 1, hasta que haya pasado la mitad del arreglo



Estructuras de Datos

Invertir Arreglo (con Métodos)

```
import * as rls from 'readline-sync';

let cantidad : number = rls.questionInt("Ingrese la cantidad de números:");

let v : number[] = new Array(cantidad);

console.log("Cargando v");
cargarVector(v, cantidad);

console.log("Mostrando invertido");
mostrarVectorInvertido(v, cantidad);

console.log("Invierto los valores en el vector");
invertirVector(v, cantidad);

console.log("Mostrando vector");
mostrarVector(v, cantidad);
```

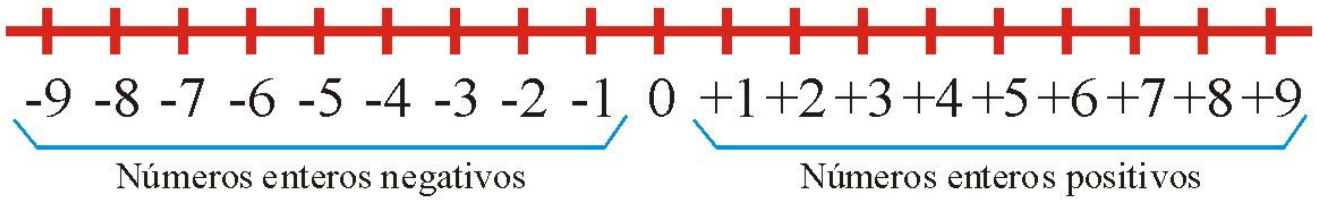


Estructuras de Datos

Tipos de Números en Arreglo (con Métodos)

- Almacene en un arreglo de dimensión N números (la cantidad es ingresada por el usuario)
- Muestre cuántos números son positivos, cuántos son negativos y cuántos ceros hay

Ejemplo:
v = 0, -7, -9, 1, 0, 0
La salida es: 1 positivos, 2 negativos y 3 ceros



Estructuras de Datos

Tipos de Números en Arreglo (con Métodos)

```
function cargarVector(v : number[], cantidad : number) {
        let indice : number;
        for (indice=0; indice<cantidad; indice++) {
                v[indice] = rls.questionInt("Ingrese el valor en ", indice, ": ");
        }
}


function mostrarVector(v : number[], cantidad : number) {
        let indice : number[];
        for (indice=0; indice<cantidad; indice++) {
                console.log(v[indice], " “);
        }
}
```

Estructuras de Datos

Tipos de Números en Arreglo (con Métodos)

```
function contarPositivos(v : number[], cantidad : number) : number  {
        let contador : number =0;
        let indice : number;

        for (indice=0; indice<cantidad; indice++) {

                if (v[indice]>0) {
                        contador++;
                }
        }
        return contador;
}
```

Estructuras de Datos

Tipos de Números en Arreglo (con Métodos)

```
function contarNegativos(v : number[], cantidad : number) : number  {
        let contador : number =0;
        let indice : number;

        for (indice=0; indice<cantidad; indice++) {

                if (v[indice]<0) {
                        contador++;
                }
        }
        return contador;
}
```

Estructuras de Datos

Tipos de Números en Arreglo (con Métodos)

```
function contarCeros(v : number[], cantidad : number) : number  {
        let contador : number =0;
        let indice : number;

        for (indice=0; indice<cantidad; indice++) {

                if (v[indice]==0) {
                        contador++;
                }
        }
        return contador;
}
```

Estructuras de Datos

Tipos de Números en Arreglo (con Métodos)

```
import * as rls from 'readline-sync';

let cantidad : number = rls.questionInt("Ingrese la cantidad de números:");

let v : number[] = new Array(cantidad);

console.log("Cargando v");
cargarVector(v, cantidad);


let numNeg = contarNegativos(v, cantidad);
let numCero = contarCeros(v, cantidad);
let numPos = contarPositivos(v, cantidad);

console.log(""Los valores de v son:");
mostrarVector(v, cantidad);

console.log(numPos, " positivos, ");
console.log(numNeg, " negativos, ");
console.log(numCero, " ceros");
```

Tratamiento de cadenas

Un dato de tipo texto puede ser visto como un arreglo de caracteres (letras, números, etc.) y por lo tanto se puede manejar de manera parcial.
Ejemplo:

```
let nombreApellido : string = 'Juan Perez';
let inicialNombre : string = nombreApellido[0];
let inicialApellido : string = nombreApellido[5];
console.log('Nombre y Apellido: ',nombreApellido,' Iniciales: ',inicialNombre,inicialApellido);
```

Tratamiento de cadenas

Un dato de tipo texto puede ser visto como un arreglo de caracteres (letras, números, etc.) y por lo tanto se puede manejar de manera parcial.
Ejemplo:

```
let nombreApellido : string = 'Juan Perez';
let inicialNombre : string = nombreApellido[0];
let inicialApellido : string = nombreApellido[5];
console.log('Nombre y Apellido: ',nombreApellido,' Iniciales: ',inicialNombre,inicialApellido);
```

Tratamiento de cadenas

En TS existen diversas funciones pre-definidas para manejar esto:

- length: método que, al igual que con arreglos,retorna la longitud de una cadena.

```
let cadena : string =”ABCDE”;
console.log(cadena.length);  //retorna 5
```

- indexOf(textoBuscado): método que retorna el índice de (o sea la posición de) la primer ocurrencia de textoBuscado en una cadena. (retorna -1 si no está, admite un 2do parámetro entero para iniciar la búsqueda desde ahí)

```
let cadena : string =”Tengo que hacer muchos ejercicios”;
console.log(cadena.indexOf(“hacer”));  //retorna 10
```

- lastIndexOf(textoBuscado): método que retorna el índice de (o sea la posición de) la última ocurrencia de textoBuscado en una cadena. (retorna -1 si no está, admite también un 2do parámetro entero para iniciar la búsqueda desde ahí)

```
let cadena : string =”Tengo que hacer y hacer y hacer muchos ejercicios”;
console.log(cadena.lastIndexOf(“hacer”));  //retorna 26
```

Tratamiento de cadenas

- substring(inicio, final) método que retorna la porción de la cadena entre las posiciones inicio y final.

```
let cadena : string =”Tengo que hacer y hacer y hacer muchos ejercicios”;
console.log(cadena.substring(10, 15));  //retorna “hacer”
```

- substr(inicio, largo) método que retorna la porción de la cadena de tamaño largo a partir de la posición inicio.

```
let cadena : string =”Tengo que hacer y hacer y hacer muchos ejercicios”;
console.log(cadena.substr(10, 21));  //retorna “hacer y hacer y hacer”
```

- toUpperCase() método que retorna una cadena con todos sus caracteres pasados a mayúsculas.

```
let cadena : string =”Tengo que hacer y hacer y hacer muchos ejercicios”;
console.log(cadena.toUpperCase());  //retorna “TENGO QUE HACER Y HACER Y HACER MUCHOS EJERCICIOS”
```

- toLowerCase() método que retorna una cadena con todos sus caracteres pasados a minúsculas.

```
let cadena : string =”Tengo que hacer y hacer y hacer muchos ejercicios”;
console.log(cadena.toLowerCase());  //retorna “tengo que hacer y hacer y hacer muchos ejercicios”
```

Tratamiento de cadenas

Conversión de tipos

- toString() método que (aplicado a un número) retorna una cadena que representa los caracteres de cada dígito.

```
let numero : number = 2021;
console.log(numero.toString());  //retorna “2021”
```

- parseInt() método que convierte una cadena con caracteres numéricos sin símbolo decimal en un número entero.

```
let cadena : string = ”2021”;
console.log(parseInt(cadena));  //retorna 2021
```

- parseFloat() método que convierte una cadena con caracteres numéricos con símbolo decimal en un número decimal.

```
let cadena : string = ”3.14159”;
console.log(parseFloat(cadena));  //retorna 3.14159
```

Técnicas de Programación

Programador full-stack

Estructuras de Datos y Métodos (Ejercicios)

Estructuras de Datos y Métodos

Multiplicación

- Implemente un método llamado “multiplicarArreglo” que recibe como parámetros tres arreglos de Enteros del mismo tamaño
- Los dos primeros arreglos contienen los números que se quieren multiplicar
- El tercer arreglo almacena el cálculo de la multiplicación de cada posición de los dos arreglos
- Utilice este método para multiplicar los siguientes cuatro arreglos de tres elementos

v1: [1, 2, 3]
v2: [4, 5, 6]
v3: [2, 1, 2]
v4: [1, 2, 1]

vResultado (v1*v2*v3*v4): [8, 20, 36]

aproveche las ventajas de métodos para resolver el ejercicio