

# Técnicas de Programación

## Carrera Programador full-stack

*Modularización y Métodos (Repaso)*

# Métodos

## *Repaso*

- **Agrupan** un conjunto de sentencias de código **cohesivas**
- Tienen un **nombre representativo**
- Pueden ser invocados
- Pueden declarar parámetros
- Pueden devolver un valor
- Nos ayudan a **reusar** el código




# Métodos

## *Repaso*

- Cada vez que se encuentra una llamada a un **método**:
  - El programa ejecuta el código del método hasta que termina
  - Vuelve a la siguiente línea del lugar donde partió

```
if (opcionMenu==1) {  
  dibujar40Guiones();  
  console.log("El resultado de la operacion es: ",  
numero1+numero2);  
}
```



```
function dibujar40Guiones() {  
  let i:number;  
  let linea:string ="";  
  for (i=1; i<=40; i++) {  
    linea=linea+"-";  
  }  
  console.log(linea);  
}
```

# Métodos con Retorno

## *Sintaxis*

```
function nombre_del_metodo(argumento_1:tipo,argumento_2:tipo,... ):tipo {  
  let retorno:tipo;  
  accion 1  
  accion 2  
  ...  
  accion n  
  
  return retorno;  
}
```

# Naming conventions

- Variables:
  - se nombran con sustantivos
- Funciones:
  - Comienzan con verbos
  - En el caso de funciones booleanas, se recomienda comenzar con "is", ej: isValid(), isAdmin(), etc...
- **usar nombres descriptivos**
  - nunca son demasiado largos!
- Evitar nombres sin significado como "aux" y "temp"

# Técnicas de Programación

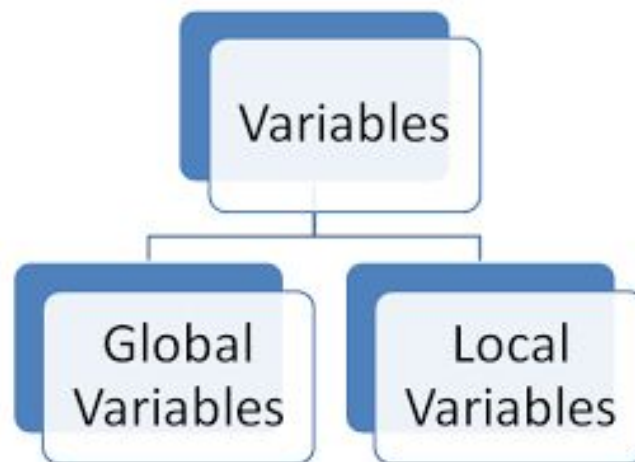
## Carrera Programador full-stack

*Ámbito de las Variables (Concepto)*

# Ámbito de las Variables

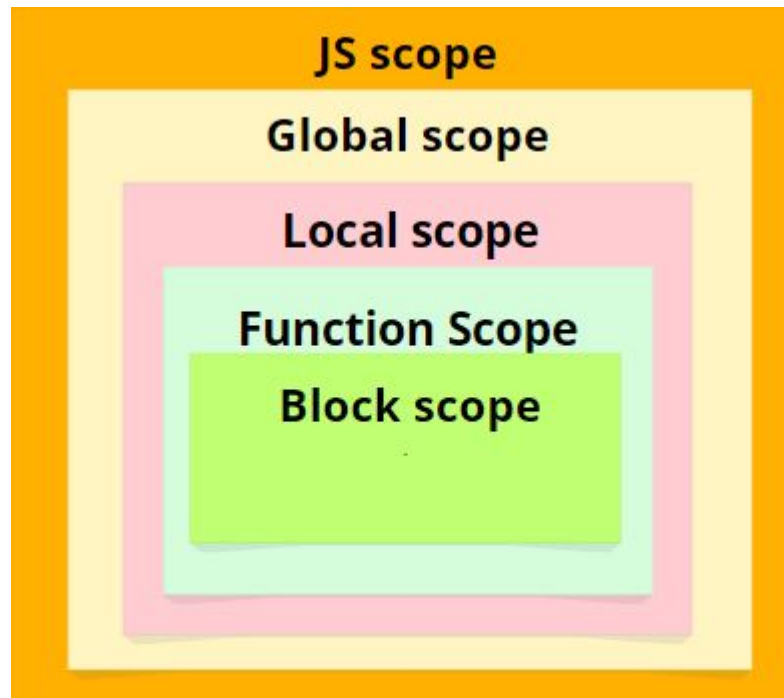
Al utilizar funciones se establece un límite para el alcance de las variables

- **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



# Ámbito de las Variables

- Se debe intentar crear métodos con variables locales y pocos parámetros para favorecer la reutilización y el mantenimiento del software





# Ámbito de las Variables

## *Ejemplos*

```
let mensaje:string = 'Hola Global!!';
```

```
ambitoVariables();
```

```
function ambitoVariables() {
```

```
    let mensaje:string;
```

```
    mensaje = 'Hola Mundo!!';
```

```
    console.log(mensaje);
```

```
}
```

# Ámbito de las Variables

## *Ejemplos*

```
let mensaje:string = 'Hola Global!!!';
```

```
ambitoVariables();
```

```
function ambitoVariables() {
```

```
    let mensaje:string;
```

```
    mensaje = 'Hola Mundo!!!';
```

```
    console.log(mensaje);
```

```
}
```

La variable local  
esconde la global

```
TS let mensaje:string = 'Hola Global!!!'; Untitled-1
1  let mensaje:string = 'Hola Global!!!';
2  ambitoVariables();
3
4  function ambitoVariables() {
5      let mensaje:string;
6      mensaje = 'Hola Mundo!!!';
7      console.log(mensaje); Hola Mundo!!
8  }
9  |
```

Si corremos este archivo con  
ts-node [nombre archivo]  
veremos en consola

# Ámbito de las Variables

## *Ejemplos*

```
let mensaje:string = 'Hola Global!!!';
```

```
ambitoVariables();
```

```
function ambitoVariables() {
```

```
    let mensaje:string;
```

```
    mensaje = 'Hola Mundo!!!';
```

```
    console.log(mensaje);
```

```
}
```

La variable local  
esconde la global

```
TS let mensaje:string = 'Hola Global!!!'; Untitled-1
1  let mensaje:string = 'Hola Global!!!';
2  ambitoVariables();
3
4  function ambitoVariables() {
5      let mensaje:string;
6      mensaje = 'Hola Mundo!!!';
7      console.log(mensaje); Hola Mundo!!
8  }
9  |
```

Si corremos este archivo con  
ts-node [nombre archivo]  
veremos en consola

# Ámbito de las Variables

## *Ejemplos*

```
ambitoVariables();
```

```
function ambitoVariables() {  
  let mensaje:string;  
  mensaje = 'Hola Mundo!!';  
  console.log(mensaje);  
}
```

---

```
ambitoVariables();
```

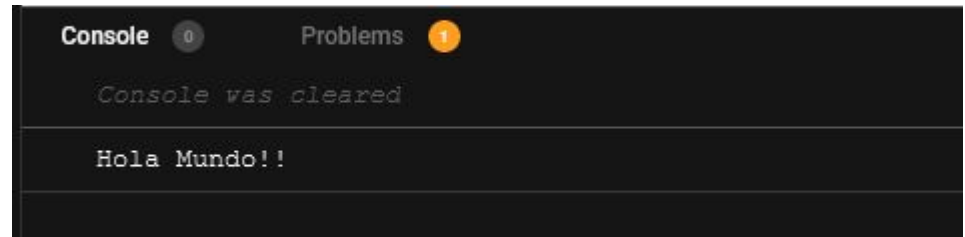
```
function ambitoVariables() {  
  leeVariable();  
}  
function leeVariable() {  
  let mensaje:string;  
  mensaje = 'Hola Mundo!!';  
  console.log(mensaje);  
}
```

# Ámbito de las Variables

## *Ejemplos*

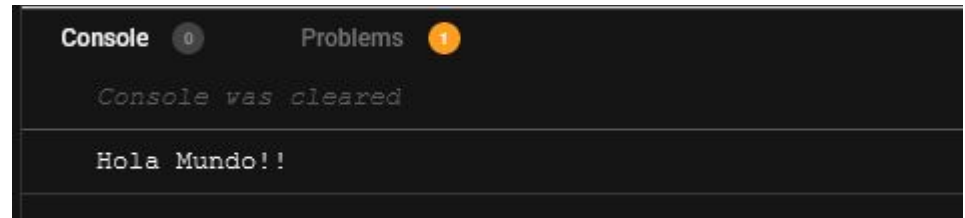
```
ambitoVariables();
```

```
function ambitoVariables() {  
  let mensaje:string;  
  mensaje = 'Hola Mundo!!';  
  console.log(mensaje);  
}
```



```
ambitoVariables();
```

```
function ambitoVariables() {  
  leeVariable();  
}  
function leeVariable() {  
  let mensaje:string;  
  mensaje = 'Hola Mundo!!';  
  console.log(mensaje);  
}
```



# Ámbito de las Variables

## *Ejemplos*

```
let mensaje:string;  
ambitoVariables();
```

```
function ambitoVariables() {  
    mensaje = 'Hola Mundo!!';  
    console.log(mensaje);  
}
```

```
let mensaje:string;  
ambitoVariables();
```

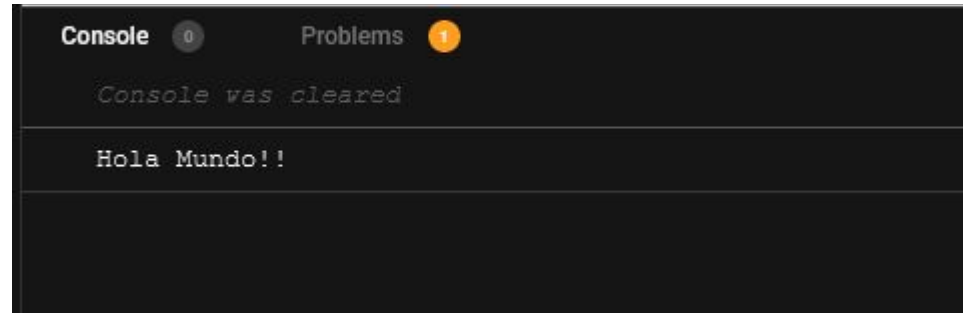
```
function ambitoVariables() {  
    leeVariable();  
}  
function leeVariable() {  
    mensaje = 'Hola Mundo!!';  
    console.log(mensaje);  
}
```

# Ámbito de las Variables

## *Ejemplos*

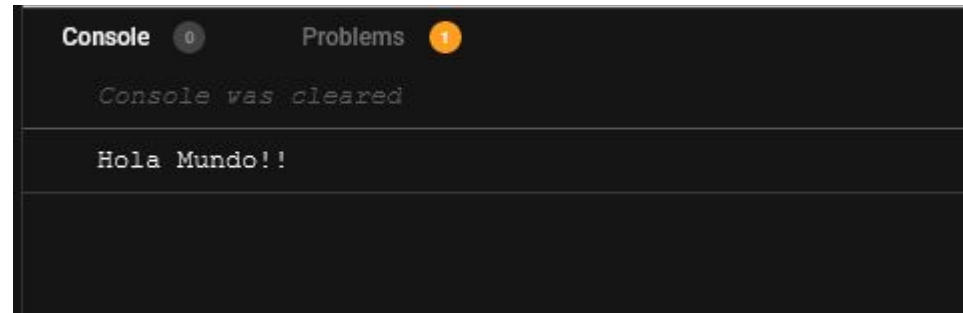
```
let mensaje:string;  
ambitoVariables();
```

```
function ambitoVariables() {  
    mensaje = 'Hola Mundo!!';  
    console.log(mensaje);  
}
```



```
let mensaje:string;  
ambitoVariables();
```

```
function ambitoVariables() {  
    leeVariable();  
}  
function leeVariable() {  
    mensaje = 'Hola Mundo!!';  
    console.log(mensaje);  
}
```



# Ámbito de las Variables

## *Ejemplos*

```
let mensaje:string;  
ambitoVariables();
```

```
function ambitoVariables() {  
    mensaje = 'Hola Mundo!!';  
    leeVariable();  
}  
function leeVariable() {  
    console.log(mensaje);  
}
```

```
let mensaje:string;  
ambitoVariables();
```

```
function ambitoVariables() {  
    leeVariable();  
    mensaje = 'Hola Mundo!!';  
}  
function leeVariable() {  
    console.log(mensaje);  
}
```



# Ámbito de las Variables

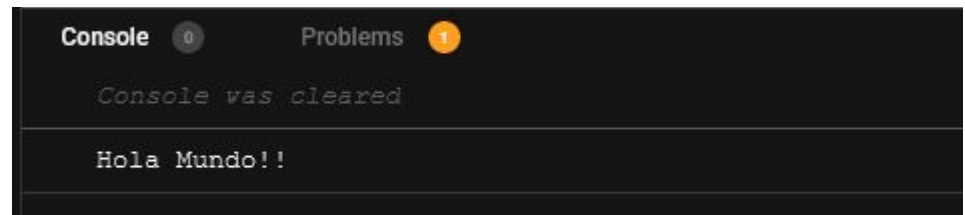
## Ejemplos

```
let mensaje:string;  
ambitoVariables();
```

```
function ambitoVariables() {  
    mensaje = 'Hola Mundo!!';  
    leeVariable();  
}  
function leeVariable() {  
    console.log(mensaje);  
}
```

```
let mensaje:string;  
ambitoVariables();
```

```
function ambitoVariables() {  
    leeVariable();  
    mensaje = 'Hola Mundo!!';  
}  
function leeVariable() {  
    console.log(mensaje);  
}
```



```
Console 0 Problems 1  
Console was cleared  
Hola Mundo!!
```



```
Console 0 Problems 2  
Console was cleared  
undefined
```

Qué pasa aquí?? No se asigna un *string* a la variable existente `mensaje`. En lugar de ello, se crea una nueva variable `mensaje` en el ámbito de la función, aunque no hayamos indicado `let`, `const` o `var`.


**No hay conflicto?? No, porque las variables “viven” en diferentes vecindarios (ámbitos)**

# Técnicas de Programación

## Carrera Programador full-stack

*Buenas Prácticas de Programación (Concepto)*

# Buenas Prácticas de Programación

- Entender el problema, diseñar una estrategia, implementar
  - Nombres representativos de variables y métodos
  - Código claro, comprensible, etc.
  - Indentación en las estructuras de control
  - Comentarios en el código
- 
- *//Así se comenta en TypeScript, con las dos barras*
  - */\* o así, esto permite un comentario de bloque \*/*

# Buenas Prácticas de Programación

- usar métodos
- No duplicar código
- Dividir el problema en subproblemas
- Construir el código tan simple como sea posible
- Que el código funcione no significa que esté bien programado. usualmente, podemos mejorarlo.



# Técnicas de Programación

## Carrera Programador full-stack

*Arreglos (Conceptos)*

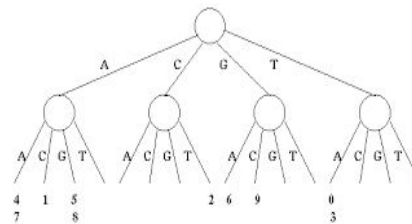
# Estructuras de Datos

## Forma particular de organizar datos



- Estructuras que permiten **CoLECCIoNAR** elementos

- GuARDARLoS
- RECoRRERLoS
- MANIPuLARLoS

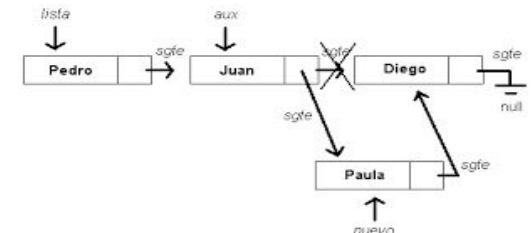


- Estructuras

- LISTAS
- CoLAS
- PILAS
- ÁRBOLES

- operaciones básicas

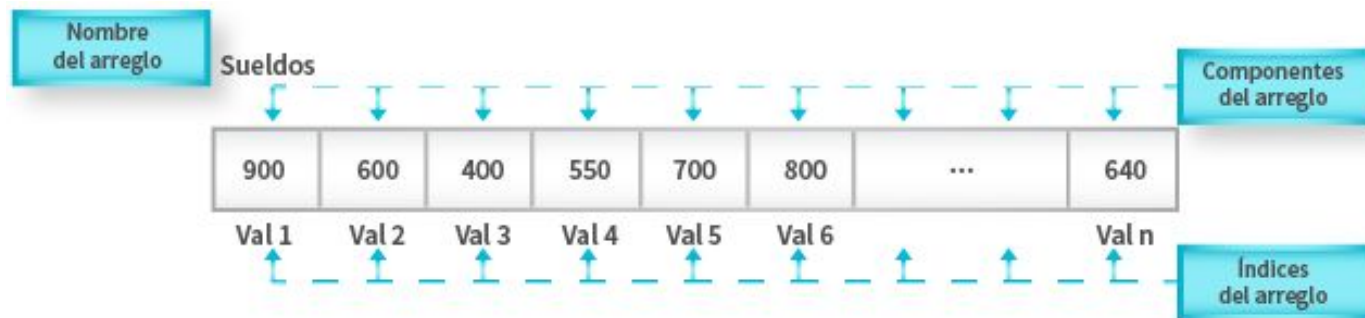
- **CoLoCAR**
- **oBTENER**



# Estructuras de Datos

## *Arreglos/Listas/Vectores*

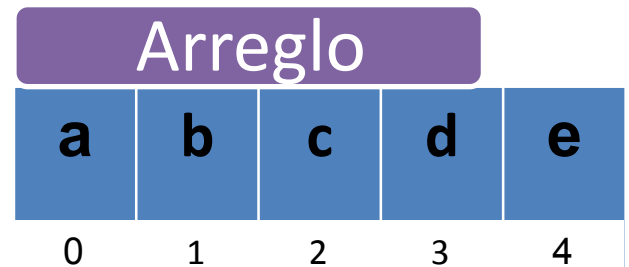
- Los arreglos son estructuras de datos homogéneas (todos sus datos son del mismo tipo)
- Permiten almacenar un determinado número de datos
- Tiene muchos elementos, y a cada uno de ellos se acceden indicando que posición se quiere usar (un índice)



# Estructuras de Datos

## *Arreglos/Listas/Vectores*

- Lista = Array
- Los elementos deben ser del mismo tipo de dato
- Zero-based (arreglos de base cero) -> Índices comienzan en 0
- La cantidad de elementos total = Length será igual al numero del ultimo elemento más 1
- Propiedades:
  - ELEMENTO o ÍTEM: a, b, c, d, e
  - LONGITUD: 5
  - ÍNDICE o SUBÍNDICE: 0, 1, 2, 3, 4



**Longitud = Length= 5**



# Estructuras de Datos

## *Definición de Arreglos*

**let** <identificador> : *\*tipo\**[] = **new Array** (<max length>);

- Los arreglos se declaran con un nombre, un tipo y luego [ ]
- Esta instrucción define un arreglo con el nombre indicado en <identificador> y 1 dimension
- El parámetro indica el valor máximo de elementos.
- La cantidad de dimensiones debe ser una, y la máxima cantidad de elementos debe ser una expresión numérica positiva
- Más adelante veremos la manera de implementar arreglos de más de una dimensión

**Ejemplo:**

**let** arregloClientes : **number**[] = **new Array**(30);

# Estructuras de Datos - Arreglos

Construya un algoritmo que según el numero de mes muestre el nombre de dicho mes

¿Como se puede hacer?



# Estructuras de Datos – Arreglos

## *Ejercicio - Identificación Mes - Código*

// Algoritmo Identificación Mes

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

```
let nroMes : number = rls.questionInt("Indique el numero de mes que le interesa: ");
```

```
switch (nroMes) {
```

```
    case 1: console.log("El mes es Enero"); break;
```

```
    case 2: console.log("El mes es Febrero"); break;
```

```
    case 3: console.log("El mes es Marzo"); break;
```

```
    case 4: console.log("El mes es Abril"); break;
```

```
    case 5: console.log("El mes es Mayo"); break;
```

```
    case 6: console.log("El mes es Junio"); break;
```

```
    case 7: console.log("El mes es Julio"); break;
```

```
    case 8: console.log("El mes es Agosto"); break;
```

```
    case 9: console.log("El mes es Septiembre"); break;
```

```
    case 10: console.log("El mes es octubre"); break;
```

```
    case 11: console.log("El mes es Noviembre"); break;
```

```
    case 12: console.log("El mes es Diciembre"); break;
```

```
    default: console.log("ud no ha escrito un numero de mes valido");
```

```
}
```

1. ENERO	7. JULIO
2. FEBRERO	8. AGOSTO
3. MARZO	9. SEPTIEMBRE
4. ABRIL	10. OCTUBRE
5. MAYO	11. NOVIEMBRE
6. JUNIO	12. DICIEMBRE

# Estructuras de Datos – Arreglos

## *otras Necesidades*

- ¿Qué pasaría si en lugar de meses fueran clientes y números de clientes?
- A medida que tengo más clientes tengo que programar más **switch** / **if** ... imposible en aplicaciones reales



# Estructuras de Datos – Arreglos

## *Ejercicio - Identificación Mes*

- Modifique el código del Ejercicio Identificación mes utilizando arreglos

Longitud = Length= 12

Enero	Febrero	Marzo	Abril	Mayo	Junio	Julio
0	1	2	3	4	5	6



# Estructuras de Datos – Arreglos

## *Ejercicio - Identificación Mes – Código*

// Algoritmo Identificación Mes

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

```
let arregloMes : string[] = new Array (12) ;
```

```
arregloMes[0] = "Enero";
```

```
arregloMes[1] = "Febrero";
```

```
arregloMes[2] = "Marzo";
```

```
arregloMes[3] = "Abril";
```

```
arregloMes[4] = "Mayo";
```

```
arregloMes[5] = "Junio";
```

```
arregloMes[6] = "Julio";
```

```
arregloMes[7] = "Agosto";
```

```
arregloMes[8] = "Septiembre";
```

```
arregloMes[9] = "octubre";
```

```
arregloMes[10] = "Noviembre";
```

```
arregloMes[11] = "Diciembre";
```

un arreglo también se puede definir “por extensión” de la siguiente manera:

```
let arregloMes : string[] = [ "Enero", "Febrero", "Marzo", "Abril",  
"Mayo", "Junio", "Julio", "Agosto", "Septiembre", "octubre",  
"Noviembre", "Diciembre" ];
```

```
let nroMes : number = rls.questionInt("Indique el numero de mes que le interesa: ");
```

```
let indice : number = nroMes - 1;
```

```
console.log("El mes es ", arregloMes[indice] );
```

# Estructuras de Datos – Arreglos

## *Ejercicio - Identificación Mes – Código*

// Algoritmo Identificación Mes

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

```
let arregloMes : string[] = new Array (12) ;
```

```
arregloMes[0] = "Enero";
```

```
arregloMes[1] = "Febrero";
```

```
arregloMes[2] = "Marzo";
```

```
arregloMes[3] = "Abril";
```

```
arregloMes[4] = "Mayo";
```

```
arregloMes[5] = "Junio";
```

```
arregloMes[6] = "Julio";
```

```
arregloMes[7] = "Agosto";
```

```
arregloMes[8] = "Septiembre";
```

```
arregloMes[9] = "octubre";
```

```
arregloMes[10] = "Noviembre";
```

```
arregloMes[11] = "Diciembre";
```

```
let nroMes : number = rls.questionInt("Indique el numero de mes que le interesa: ");
```

```
let indice : number = nroMes - 1;
```

```
console.log("El mes es ", arregloMes[indice] );
```

Recuerde que al ser el arreglo en base 0 hay que restar 1 al índice, porque el usuario va a ingresar el numero de mes empezando desde 1

# Estructuras de Datos – Arreglos

## *Ejercicio – Arreglo de Números*

- Construya un algoritmo que tenga un arreglo de números y se los muestre al usuario
- El arreglo debe ser llamado num
- El arreglo num debe contener los siguientes datos: 20, 14, 8, 0, 5, 19 y 24.
- Mostrar los valores resultantes del arreglo



# Estructuras de Datos – Arreglos

## *Ejercicio – Arreglo de Números*

- Crear un arreglo llamado num que almacene los siguientes datos: 20, 14, 8, 0, 5, 19 y 24 y se los muestre al usuario
- Al utilizar arreglos en base cero los elementos válidos van de 0 a  $n-1$ , donde  $n$  es el tamaño del arreglo
- En el ejemplo 1 las posiciones/índice del num entonces van desde 0 a  $7-1$ , es decir de 0 a 6

	num						
Datos del arreglo	20	14	8	0	5	19	24
Posiciones	0	1	2	3	4	5	6

# Estructuras de Datos – Arreglos

## *Ejercicio – Arreglo de Numeros - Código*

```
// Algoritmo ArregloNumeros
```

```
let num : number[] = new Array (7) ;
```

```
let indice: number = 0;
```

```
num[0] = 20;
```

```
num[1] = 14;
```

```
num[2] = 8;
```

```
num[3] = 0;
```

```
num[4] = 5;
```

```
num[5] = 19;
```

```
num[6] = 24;
```

```
while (indice < 7) {
```

```
    console.log ("El numero en la posicion ", indice, " es ", num[indice]);
```

```
    indice++;
```

```
}
```

# Estructuras de Datos – Arreglos

## *Ejercicio – Arreglo de Numeros - Código*

```
// Algoritmo ArregloNumeros
```

```
let num : number[] = new Array (7) ;
```

```
let indice: number = 0;
```

```
num[0] = 20;
```

```
num[1] = 14;
```

```
num[2] = 8;
```

```
num[3] = 0;
```

```
num[4] = 5;
```

```
num[5] = 19;
```

```
num[6] = 24;
```

```
while (indice < 7) {
```

```
    console.log ("El numero en la posicion ", indice, "
```

```
es ", num[indice]);
```

```
    indice++;
```

```
}
```

El número en la posición	0	es	20
El número en la posición	1	es	14
El número en la posición	2	es	8
El número en la posición	3	es	0
El número en la posición	4	es	5
El número en la posición	5	es	19
El número en la posición	6	es	24

# Estructuras de Datos – Arreglos

## *Ejercicio – Números Deseados*

- Construya un algoritmo que tenga un arreglo de dimensión 5 y llénelo con los números que el usuario desee.
- Muestre los números del arreglo al usuario

# Estructuras de Datos – Arreglos

## *Ejercicio – Números Deseados - Código*

```
// Algoritmo NumerosDeseados
```

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

```
let nroDeseadoArreglo : number[] = new Array (5);
```

```
let nro : number, indice : number;
```

```
for (indice = 0; indice < 5; indice++) {
```

```
    nro = rls.questionInt(`Indique el numero que desea incorporar en la posicion ${indice}: `);
```

```
    nroDeseadoArreglo[indice] = nro;
```

```
}
```

```
for (indice = 0; indice < 5; indice++) {
```

```
    console.log(`El numero en la posicion ${indice} es ${nroDeseadoArreglo[indice]}`);
```

```
}
```

# Estructuras de Datos – Arreglos

## *Ejercicio – Números Deseados - Código*

```
// Algoritmo NumerosDeseados
```

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

```
let nroDeseadoArreglo : number[] = new Array (5);
```

```
let nro : number, indice : number;
```

```
for (indice = 0; indice < 5; indice++) {
```

```
    nro = rls.questionInt(`Indique el numero que desea incorporar en la posicion ${indice}: `);
```

```
    nroDeseadoArreglo[indice] = nro;
```

```
}
```

```
for (indice = 0; indice < 5; indice++) {
```

```
    console.log(`El numero en la posicion ${indice} es ${nroDeseadoArreglo[indice]}`);
```

```
}
```

```
El número en la posición 0 es 1
```

```
El número en la posición 1 es 2
```

```
El número en la posición 2 es 3
```

```
El número en la posición 3 es 4
```

```
El número en la posición 4 es 5
```

# Estructuras de Datos – Arreglos

## *Ejercicio – Nombres Deseados*

- Construya un algoritmo que tenga un arreglo de dimensión deseada por el usuario y llénelo con los nombres que el usuario desee
- Crear un arreglo de las posiciones que desee el usuario y llenarlo con nombres de personas

# Estructuras de Datos – Arreglos

## *Ejercicio – Nombres Deseados - Código*

// Algoritmo NombresDeseados

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

```
let dimensionArreglo : number = rls.questionInt(`Ingrese la dimension del arreglo: `);
```

```
let nombrePersonas : string[] = new Array (dimensionArreglo);
```

```
let indice : number;
```

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

```
    nombrePersonas[indice] = rls.question(`Ingrese el nombre que quiere poner en el lugar ${indice}: `);  
}
```

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

```
    console.log(`La persona que ingreso en la posicion ${indice} es: ${nombrePersonas[indice]}`);  
}
```



# Estructuras de Datos – Arreglos

## *Ejercicio – Nombres Deseados - Código*

```
// Algoritmo NombresDeseados
```

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

```
let dimensionArreglo : number = rls.questionInt(`Ingrese la dimension del arreglo: `);
```

```
let nombrePersonas : string[] = new Array (dimensionArreglo);
```

```
let indice : number;
```

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

```
    nombrePersonas[indice] = rls.question(`Ingrese el nombre que quiere poner en el lugar ${indice}: `);
```

```
}
```

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

```
    console.log(`La persona que ingreso en la posicion ${indice} es: ${nombrePersonas[indice]}`);
```

```
}
```

```
La persona que ingresó en la posición 0 es: carlos
```

```
La persona que ingresó en la posición 1 es: laura
```

```
La persona que ingresó en la posición 2 es: rene
```

# Estructuras de Datos – Arreglos

## *Ejercicio – Dos Arreglos*

- Construya un algoritmo que tenga dos arreglos uno que almacene 2 nombres y otro que almacene 3 números ambos ingresados por el usuario.
- Al final debe imprimir los valores por consola.

# Estructuras de Datos – Arreglos

## *Ejercicio – Dos Arreglos - Código*

// Algoritmo DosArreglos

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

```
let arregloNombres : string[] = new Array (2);
```

```
let arregloNumeros : number[] = new Array (3);
```

```
let indice : number;
```

```
for (indice = 0; indice < 2; indice++) {  
    arregloNombres[indice] = rls.question(`Ingrese el nombre de la posicion ${indice}: `);  
}
```

```
for (indice = 0; indice < 3; indice++) {  
    arregloNumeros[indice] = rls.questionInt(`Ingrese el numero de la posicion ${indice}: `);  
}
```

```
for (indice = 0; indice < 2; indice++) {  
    console.log(`El nombre en la posicion ${indice} es: ${arregloNombres[indice]}`);  
}
```

```
for (indice = 0; indice < 3; indice++) {  
    console.log(`El numero en la posicion ${indice} es: ${arregloNumeros[indice]}`);  
}
```

# Estructuras de Datos – Arreglos

## *Ejercicio – Dos Arreglos - Código*

// Algoritmo DosArreglos

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

```
let arregloNombres : string[] = new Array (2);
```

```
let arregloNumeros : number[] = new Array (3);
```

```
let indice : number;
```

```
for (indice = 0; indice < 2; indice++) {  
    arregloNombres[indice] = rls.question(`Ingrese el nombre de la posicion ${indice}: `);  
}
```

```
for (indice = 0; indice < 3; indice++) {  
    arregloNumeros[indice] = rls.questionInt(`Ingrese el numero de la posicion ${indice}: `);  
}
```

```
for (indice = 0; indice < 2; indice++) {  
    console.log(`El nombre en la posicion ${indice} es: ${arregloNombres[indice]}`);  
}
```

```
for (indice = 0; indice < 3; indice++) {  
    console.log(`El numero en la posicion ${indice} es: ${arregloNumeros[indice]}`);  
}
```

El nombre en la posición 0 es: oscar

El nombre en la posición 1 es: alejandra

El número en la posición 0 es: 2

El número en la posición 1 es: 4

El número en la posición 2 es: 8

# Estructuras de Datos – Arreglos

## *Ejercicio – Suma Elementos Arreglo*

- Construya un algoritmo que sume todos los elementos de un arreglo de tamaño  $N$
- La dimensión del arreglo es ingresada por el usuario
- Los elementos (números) del arreglo son ingresados por el usuario

# Estructuras de Datos – Arreglos

## *Ejercicio – Suma Elementos Arreglo - Código*

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

let dimensionArreglo : number = rls.questionInt(`Ingrese la dimension del arreglo: `);
let arreglo : number[] = new Array (dimensionArreglo);
let indice : number;
let resultado : number = 0;
for (indice = 0; indice < dimensionArreglo; indice++) {
    arreglo[indice] = rls.questionInt(`Indique el nro que va en la posicion ${indice}: `);
    resultado += arreglo[indice];
}
for (indice = 0; indice < dimensionArreglo; indice++) {
    console.log(`El numero en la posicion ${indice} es: ${arreglo[indice]}`);
}
console.log(`La suma del arreglo es: ${resultado}`);
```

# Estructuras de Datos – Arreglos

## *Ejercicio – Suma Elementos Arreglo - Código*

// Algoritmo SumaElementosArreglo

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

```
let dimensionArreglo : number = rls.questionInt(`Ingrese la dimension del arreglo: `);
let arreglo : number[] = new Array (dimensionArreglo);
let indice : number;
let resultado : number = 0;
for (indice = 0; indice < dimensionArreglo; indice++) {
    arreglo[indice] = rls.questionInt(`Indique el nro que va en la posicion ${indice}: `);
    resultado += arreglo[indice];
}
for (indice = 0; indice < dimensionArreglo; indice++) {
    console.log(`El numero en la posicion ${indice} es: ${arreglo[indice]}`);
}
console.log(`La suma del arreglo es: ${resultado}`);
```

```
C:\cursos\cfs\arreglos>ts-node arrayReduce
Ingrese la dimension del arreglo: 5
Indique el nro que va en la posicion 0: 1
Indique el nro que va en la posicion 1: 2
Indique el nro que va en la posicion 2: 3
Indique el nro que va en la posicion 3: 4
Indique el nro que va en la posicion 4: 5
El nro en la posicion 0 es: 1
El nro en la posicion 1 es: 2
El nro en la posicion 2 es: 3
El nro en la posicion 3 es: 4
El nro en la posicion 4 es: 5
La suma del arreglo es: 15
```

# Estructuras de Datos – Arreglos

## *Ejercicio – Completar Arreglo*

- Llenar un array de 10 posiciones con números aleatorios entre 0 y 99
- Para obtener números aleatorios crear una función Azar, que se base en las funciones disponibles en el paquete Math:
  - Math.random() devuelve un nro al azar entre 0 y 1.



# Estructuras de Datos – Arreglos

## *Ejercicio – Completar Arreglo - Código*

//Algoritmo CompletarArreglo

```
let arregloCompletar : number[] = new Array (10);
```

```
let indice : number;
```

```
for (indice = 0; indice < 10; indice++) {
```

```
    arregloCompletar[indice] = Azar(100);
```

```
}
```

```
for (indice = 0; indice < 10; indice++) {
```

```
    console.log (`El numero en la posicion ${indice} es: ${arregloCompletar[indice]}`);
```

```
}
```

```
function Azar (tope : number) : number {
```

```
    return Math.floor(Math.random()*tope);
```

```
};
```

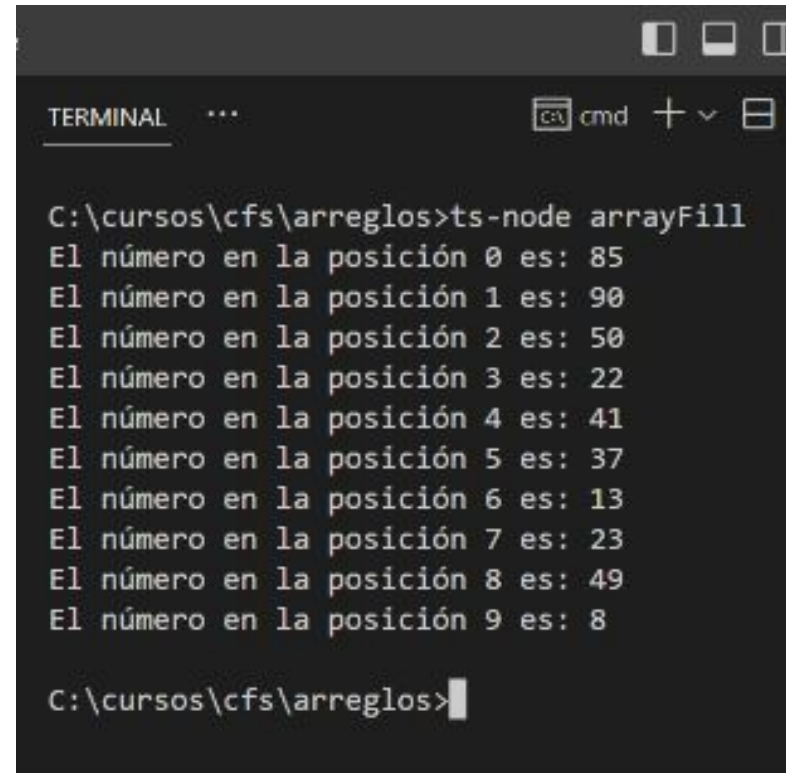
# Estructuras de Datos – Arreglos

## *Ejercicio – Completar Arreglo - Código*

//Algoritmo CompletarArreglo

```
let arregloCompletar : number[] = new Array (10);
let indice : number;
for (indice = 0; indice < 10; indice++) {
    arregloCompletar[indice] = Azar(100);
}
for (indice = 0; indice < 10; indice++) {
    console.log (`El numero en la posicion ${indice} es:
${arregloCompletar[indice]}`);
}

function Azar (tope : number) : number {
    return Math.floor(Math.random()*tope);
};
```



```
TERMINAL ... cmd + v [icon]

C:\cursos\cfs\arreglos>ts-node arrayFill
El número en la posición 0 es: 85
El número en la posición 1 es: 90
El número en la posición 2 es: 50
El número en la posición 3 es: 22
El número en la posición 4 es: 41
El número en la posición 5 es: 37
El número en la posición 6 es: 13
El número en la posición 7 es: 23
El número en la posición 8 es: 49
El número en la posición 9 es: 8

C:\cursos\cfs\arreglos>
```

# Técnicas de Programacion

## Carrera Programador full-stack

*Arreglos (Ejercicios)*

# Estructuras de Datos

## *Crear arreglo*

- 1) Crear un arreglo de letras e imprimirlo
- 2) Dado un array con nombres de tamaño 5, pedir al usuario que ingrese un nombre y verificar si está en el arreglo. Imprimir el arreglo y si está o no en él.

# Estructuras de Datos

*Encontrar el elemento más grande del arreglo*

Dado el siguiente arreglo

[4,7,9,3,1,45,67,23,29,78,11,16]

- Crear un programa que encuentre cuál es el número más grande del arreglo e imprimirlo por consola
- Almacenar el número más grande en una variable global y pasarlo a una función para determinar si el número es par o impar

# Estructuras de Datos

## *Sumar Dos Arreglos*

- Sumar los elementos de cada una de las posiciones de dos arreglos y guardar el resultado en otro arreglo
- 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.

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

$$\begin{aligned} A+B &= \langle (5+17), (9+(-3)), ((-10)+(-2)) \rangle \\ &= \langle 22, 6, -12 \rangle \end{aligned}$$

# 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 ultimo 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

