

Técnicas de Programación

Programador full-stack

Modularización y Métodos (Repaso)

Métodos

Repaso

- **Agrupar** 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;  
  acción 1  
  acción 2  
  ...  
  acción n  
  
  return retorno;  
}
```

Técnicas de Programación

Programador full-stack

Modularización y Métodos (Ejercicios)

Métodos

Ejercicio: Potencias

- Realice un programa que devuelva la potencia de un número.
- La base y el exponente deben ser ingresados por teclado.
- Sólo deben admitirse exponentes mayores o iguales a cero.
- Recuerde que el resultado de un numero elevado a 0 es 1.
- Separe la lógica de calcular la potencia utilizando métodos.

The diagram illustrates the calculation of a power. It shows the equation $3^4 = 3 \times 3 \times 3 \times 3 = 81$. Three labels with arrows point to specific parts of the equation: 'Base' (in blue) points to the number 3, 'Exponente' (in orange) points to the superscript 4, and 'Resultado' (in purple) points to the final value 81.

Métodos

Ejercicio: Múltiplos

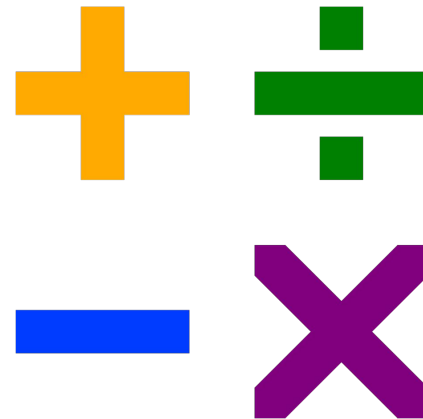
- Cree un método esMultiplo con 2 parámetros que devuelva el valor lógico **verdadero** o **falso** según si el primer número que se indique como parámetro es múltiplo del segundo
- Recuerde que un numero es múltiplo de otro si al dividirlo su resto es cero
- Recuerde que la operación **mod** permite saber si el resto de una división es cero

dividendo		divisor
40		8
resto - 0		5
		cociente

Métodos

Ejercicio: Divisores

- Implemente un método llamado `cantidadDeDivisores` que reciba un número entero y devuelva la cantidad de divisores
- Por ejemplo, para el número 16, sus divisores son 1, 2, 4, 8, 16, por lo que la respuesta debería ser 5
- **Re-utilice** el método `esMultiplo` implementado para el ejercicio anterior



Técnicas de Programación

Programador full-stack

Modularización y Métodos (Resolución)

Métodos

Ejercicio: Potencias

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

```
let base : number = rls.questionInt("ingrese Base: ");
```

```
let exponente : number = rls.questionInt("ingrese Exponente: ");
```

```
console.log("La potencia es = ", calcularPotencia(base, exponente));
```

```
function calcularPotencia (base : number, exponente : number) : number {
```

```
    let numero : number = 1;
```

```
    if (exponente == 0) {
```

```
        return 1;
```

```
    } else {
```

```
        for (let i = 1; i <= exponente; i++) {
```

```
            numero = numero * base;
```

```
        };
```

```
        return numero;
```

```
    }
```

```
};
```

```
function calcularPotencia (base:number, exponente:number) : number {
```

```
    return Math.pow (base, exponente);
```

```
};
```

Métodos

Ejercicio: Múltiplos

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

```
let numero1 : number = rls.questionInt("Indique el número a verificar: ");
```

```
let numero2 : number = rls.questionInt("Indique el número múltiplo: ");
```

```
if ( esMultiplo (numero1, numero2) ) {
```

```
    console.log (numero1, "ES número múltiplo de ", numero2);
```

```
} else {
```

```
    console.log (numero1, "NO es número múltiplo de ", numero2);
```

```
}
```

```
function esMultiplo (numero1 : number, numero2 : number) : boolean {
```

```
    return ((numero1 % numero2) == 0 );
```

```
};
```

Métodos

Ejercicio: Divisores

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

```
let numero : number = rls.questionInt("Indique un número: ");
```

```
let cantDivisores : number = contarDivisores( numero );
```

```
console.log("El número", numero, "tiene ", cantDivisores, "divisores.");
```

```
function contarDivisores ( numero : number ) : number {
```

```
    let cantidad : number = 0;
```

```
    let divisor : number;
```

```
    for (divisor = 1; divisor <= numero; divisor++) {
```

```
        if (esMultiplo (numero, divisor)) {
```

```
            cantidad++;
```

```
        }
```

```
    }
```

```
    return cantidad;
```

```
};
```

```
function esMultiplo (numero1 : number, numero2 : number) : boolean {
```

```
    return ((numero1 % numero2) == 0);
```

```
};
```

Técnicas de Programación

Programador full-stack

Ámbito de las Variables (Concepto)

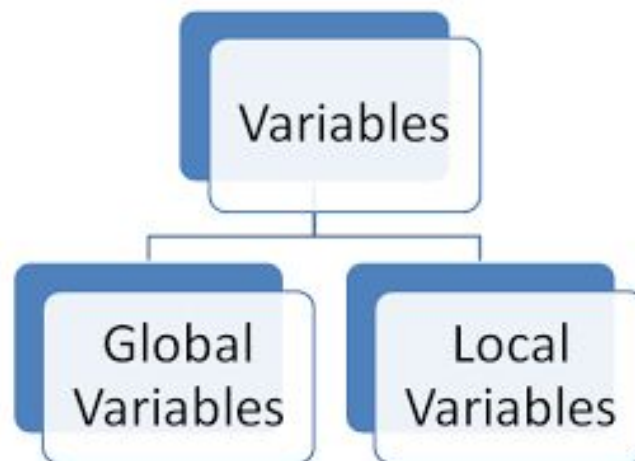
Naming

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

Ámbito de las Variables

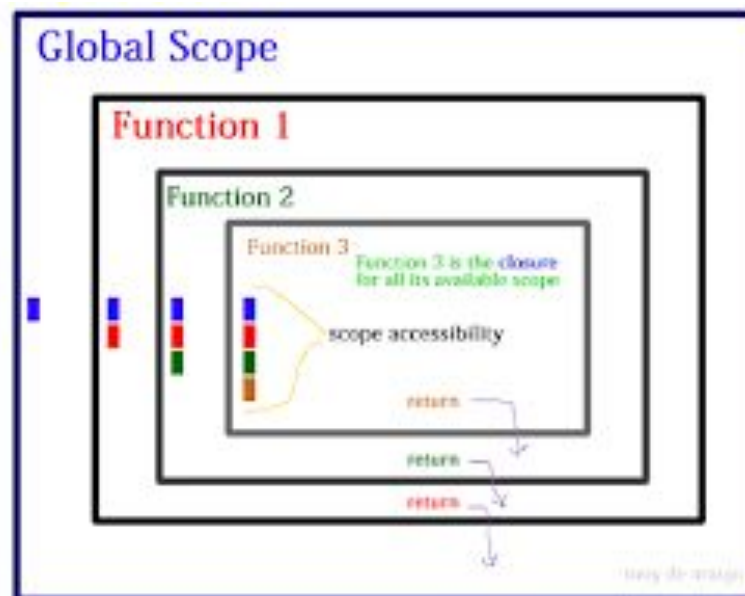
Al utilizar funciones se establece un límite para el alcance de las variables

- **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);  
}
```

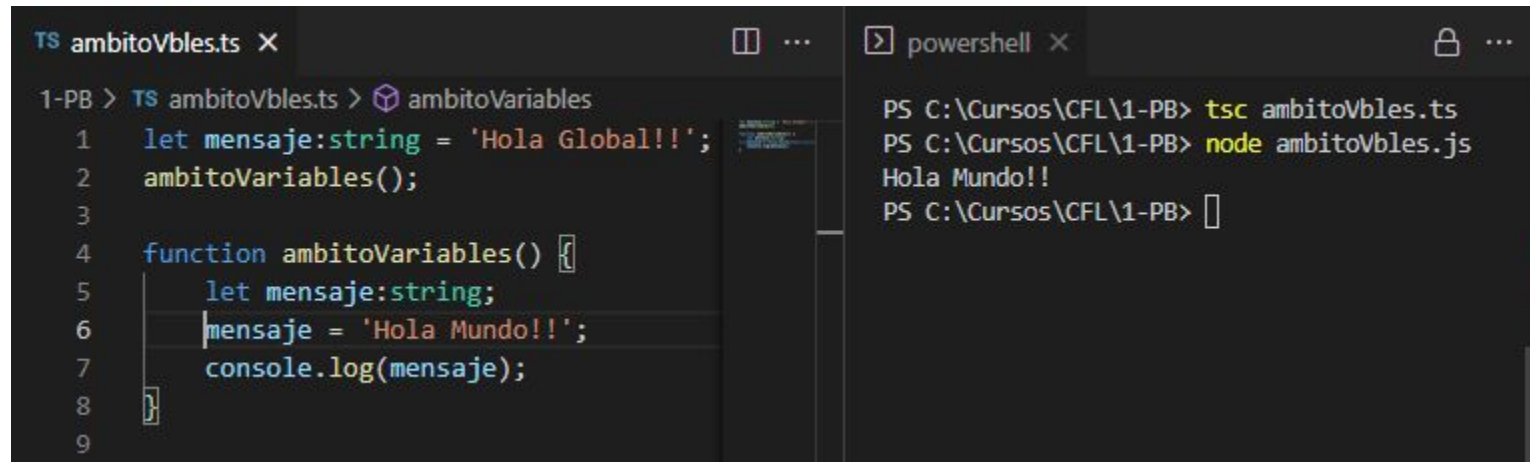
Ámbito de las Variables

Ejemplos

```
let mensaje:string = 'Hola Global!!!';  
ambitoVariables();
```

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

La variable local
esconde la global



The screenshot shows a code editor on the left and a PowerShell terminal on the right. The code editor displays the TypeScript code from the slide, with line numbers 1 through 9. The PowerShell terminal shows the commands to compile and run the code, resulting in the output 'Hola Mundo!!'.

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

```
PS C:\Cursos\CFL\1-PB> tsc ambitoVbles.ts  
PS C:\Cursos\CFL\1-PB> node ambitoVbles.js  
Hola Mundo!!  
PS C:\Cursos\CFL\1-PB>
```

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);
}
```

```
TS ambitoVbles.ts X
1-PB > TS ambitoVbles.ts > ...
1  ambitoVariables();
2
3  function ambitoVariables() {
4    let mensaje:string;
5    mensaje = 'Hola Mundo!!';
6    console.log(mensaje);
7  }
8

powershell X
PS C:\Cursos\CFL\1-PB> tsc ambitoVbles.ts
PS C:\Cursos\CFL\1-PB> node ambitoVbles.js
Hola Mundo!!
PS C:\Cursos\CFL\1-PB> 
```

ambitoVariables();

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

```
TS ambitoVbles.ts X
1-PB > TS ambitoVbles.ts > leeVariable
1  ambitoVariables();
2
3  function ambitoVariables() {
4    leeVariable();
5  }
6  function leeVariable() {
7    let mensaje:string;
8    mensaje = 'Hola Mundo!!';
9    console.log(mensaje);
10 }
11

powershell X
PS C:\Cursos\CFL\1-PB> tsc ambitoVbles.ts
PS C:\Cursos\CFL\1-PB> node ambitoVbles.js
Hola Mundo!!
PS C:\Cursos\CFL\1-PB> 
```

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);
}
```

```
TS ambitoVbles.ts x
1-PB > TS ambitoVbles.ts > ambitoVariables
1 let mensaje:string;
2 ambitoVariables();
3
4 function ambitoVariables() {
5     let mensaje:string;
6     mensaje = 'Hola Mundo!!';
7     console.log(mensaje);
8 }
9
```

```
PS C:\Cursos\CFL\1-PB> tsc ambitoVbles.ts
PS C:\Cursos\CFL\1-PB> node ambitoVbles.js
Hola Mundo!!
PS C:\Cursos\CFL\1-PB>
```

```
TS ambitoVbles.ts x
1-PB > TS ambitoVbles.ts > ...
1 let mensaje:string;
2 ambitoVariables();
3
4 function ambitoVariables() {
5     leeVariable();
6 }
7 function leeVariable() {
8     let mensaje:string;
9     mensaje = 'Hola Mundo!!';
10    console.log(mensaje);
11 }
12
```

```
PS C:\Cursos\CFL\1-PB> tsc ambitoVbles.ts
PS C:\Cursos\CFL\1-PB> node ambitoVbles.js
Hola Mundo!!
PS C:\Cursos\CFL\1-PB>
```

Ámbito de las Variables

Ejemplos

```
let mensaje;  
ambitoVariables();
```

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

```
let mensaje;  
ambitoVariables();
```

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

Ámbito de las Variables

Ejemplos

```
let mensaje;
ambitoVariables();
```

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

```
let mensaje;
ambitoVariables();
```

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

The screenshot shows a VS Code editor with a file named `ambitoVbles.ts`. The code defines a `mensaje` variable and a function `ambitoVariables` that calls `leeVariable`. The `leeVariable` function logs the value of `mensaje`. To the right, a PowerShell terminal shows the command `tsc ambitoVbles.ts` being executed, followed by `node ambitoVbles.js`, which outputs `Hola Mundo!!`.

```
TS ambitoVbles.ts x
1-PB > TS ambitoVbles.ts > ...
1 let mensaje;
2 ambitoVariables();
3
4 function ambitoVariables() {
5     mensaje = "Hola Mundo!!";
6     leeVariable();
7 }
8 function leeVariable() {
9     console.log(mensaje);
10 }
11
```

```
PS C:\Cursos\CFL\1-PB> tsc ambitoVbles.ts
PS C:\Cursos\CFL\1-PB> node ambitoVbles.js
Hola Mundo!!
PS C:\Cursos\CFL\1-PB>
```

The screenshot shows the same VS Code editor with the same TypeScript code as the first example. However, in the PowerShell terminal, the output of `node ambitoVbles.js` is `undefined`. This is because in this version of the code, the variable `mensaje` is only assigned inside the `ambitoVariables` function, and it is not yet assigned when `leeVariable` is called.

```
TS ambitoVbles.ts x
1-PB > TS ambitoVbles.ts > ambitoVariables
1 let mensaje;
2 ambitoVariables();
3
4 function ambitoVariables() {
5     leeVariable();
6     mensaje = "Hola Mundo!!";
7 }
8 function leeVariable() {
9     console.log(mensaje);
10 }
11
```

```
PS C:\Cursos\CFL\1-PB> tsc ambitoVbles.ts
PS C:\Cursos\CFL\1-PB> node ambitoVbles.js
undefined
PS C:\Cursos\CFL\1-PB>
```


Técnicas de Programación

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*



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



Técnicas de Programación

Programador full-stack

Arreglos (Conceptos)

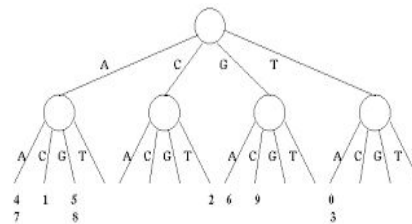
Estructuras de Datos

Forma particular de organizar datos



- Estructuras que permiten **COLECCIONAR** elementos

- GUARDARLOS
- RECORRERLOS
- MANIPULARLOS

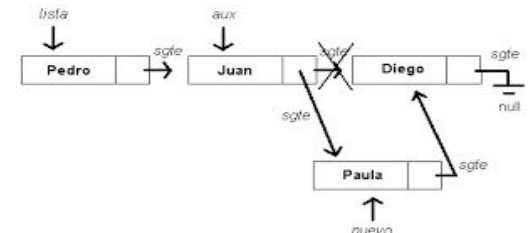


- Estructuras

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

- Operaciones básicas

- **COLOCAR**
- **OBTENER**



Estructuras de Datos - Arreglos

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

¿Cómo 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 número 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 número de mes válido");
```

```
}
```

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

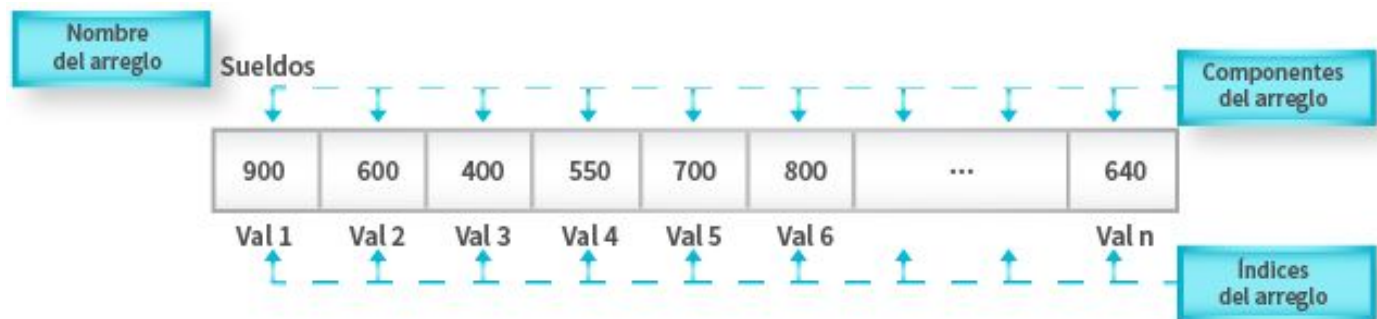
- ¿Qué pasa si en lugar de meses fueran clientes y números de clientes?
- A medida que tengo más clientes tengo que programar más **switch** / **if** ... imposible en aplicaciones reales



Estructuras de Datos

Arreglos/Listas/Vectores

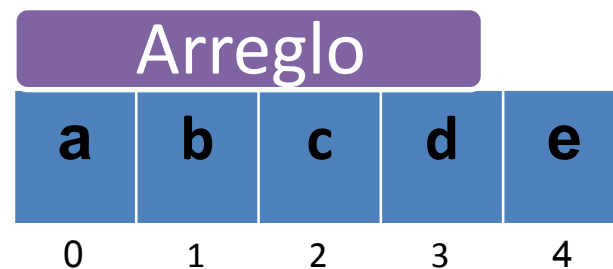
- Los arreglos son estructuras de datos homogéneas (todos sus datos son del mismo tipo)
- Permiten almacenar un determinado número de datos
- Tiene muchos elementos, y a cada uno de ellos se acceden indicando que posición se quiere usar (un índice)



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 número del último elemento más 1
- Propiedades:
 - ELEMENTO o ITEM: a, b, c, d, e
 - LONGITUD: 5
 - INDICE o SUBINDICE: 0, 1, 2, 3, 4



Longitud = Length = 5

Estructuras de Datos

Definición de Arreglos

let <identificador> : **tipo** [] = **new Array** (<maxI>);

- Los arreglos se declaran con un nombre, un tipo y luego []
- Esta instrucción define un arreglo con el nombre indicado en <identificador> y 1 dimensión
- 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 entera y 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

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 tambien 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 número 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 número de mes que le interesa: ");
```

```
let indice = 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 número 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 validos van de 0 a $n-1$, donde n es el tamaño del arreglo
- En el ejemplo 1 las posiciones/indice 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 – 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 – 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 – 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 – 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 – 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.

Matrices

- Permiten representar más de 1 dimensión (a diferencia de los arreglos)
- Si tienen 2 dimensiones, son como tablas (n filas y m columnas)
- Si tienen 3 dimensiones, son como espacios con ancho, alto y profundidad (X, Y, Z)
- En TypeScript no existen los arreglos multidimensionales, estos se definen anidando arreglos dentro de arreglos.

$$A = \begin{bmatrix} b_{11} & b_{12} & b_{13} & \dots & b_{1n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ b_{i1} & b_{i2} & b_{i3} & \dots & b_{in} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & b_{m3} & \dots & b_{mn} \end{bmatrix}$$

n Columnas

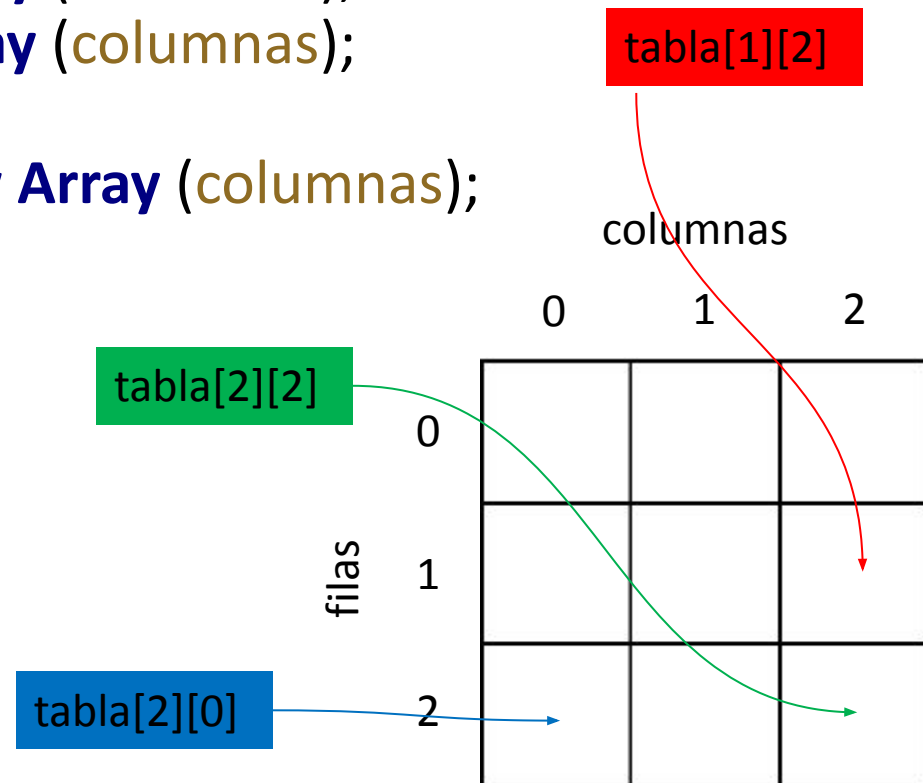
m Filas

Matrices

```
identificador : tipo[] = new Array (filas);  
identificador[0] : tipo[] = new Array (columnas);  
identificador[1] : tipo[] = new Array (columnas);  
...  
identificador[filas-1] : tipo[] = new Array (columnas);
```

• *Ejemplo:* tabla (3,3)

```
tabla : tipo[] = new Array (3);  
tabla[0] : tipo[] = new Array (3);  
tabla[1] : tipo[] = new Array (3);  
tabla[2] : tipo[] = new Array (3);
```



Matrices

Para recorrer una matriz necesitamos 2 índices

```
let fila : number, columna : number;  
for (fila = 0 ; fila < nroFilas ; fila++) {  
    for (columna = 0 ; columna < nroColumnas ; columna++) {  
        console.log (matriz[fila][columna], " ");  
    }  
}
```


Técnicas de Programación

Programador full-stack

Arreglos (Ejercicios)

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 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

