

Técnicas de Programación

Programador full-stack

Repaso General para el Examen

Técnicas de Programación

Programador full-stack

Estructuras de Datos y Métodos (Ejercicios)

Estructuras de Datos y Métodos

Cadenas

Ejercicio: Convertir una palabra ingresada por el usuario en clave, según las reglas siguientes:

- si el carácter es una vocal reemplazar aeiou por . , ; : ! respectivamente.
- si el carácter es un número o un operador matemático (+ - * /) queda igual.
- si el carácter es una consonante minúscula pasar a mayúscula y viceversa.

MauriCio2781* → *m.!R;c;:2781

Estructuras de Datos y Métodos

Cadenas

- Solicite al usuario que ingrese un texto y retórnelo convertido en un nombre de variable/función con las reglas camelCase

- Por ejemplo, si el usuario ingresa:

convertir texto según camel case

el programa lo debe convertir en:

convertirTextoSegunCamelCase

Estructuras de Datos y Métodos

Producto Escalar

- Cargue dos arreglos de dimensión N números (la cantidad es ingresada por el usuario)
- Calcule el producto escalar entre los dos arreglos:

Si $A < a_1, b_1, c_1 >$ y $B < a_2, b_2, c_2 >$

El producto escalar entre A y B en función de sus componentes está dado por:

$$A \cdot B = a_1 a_2 + b_1 b_2 + c_1 c_2$$

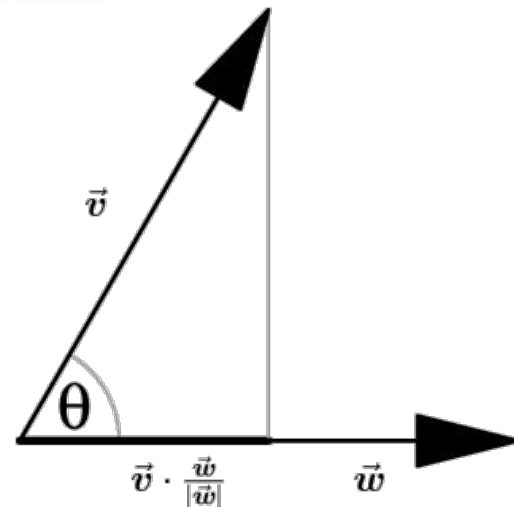
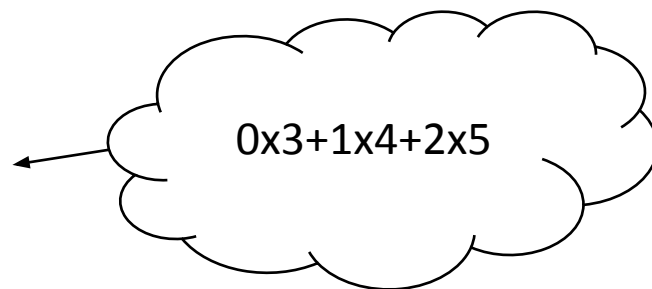
Ejemplo:

n = 3

v1 = 0, 1, 2

v2 = 3, 4, 5

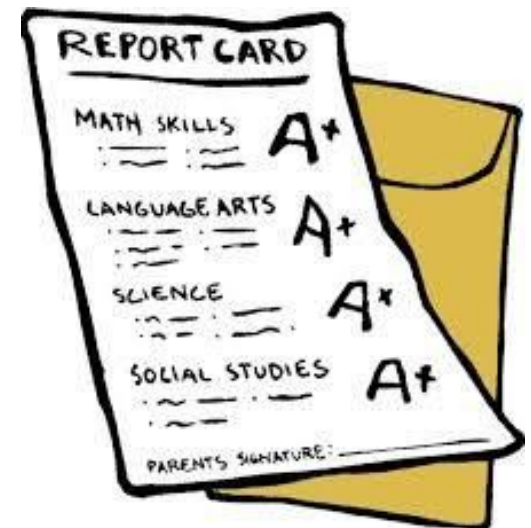
producto = 14



Estructuras de Datos y Métodos

Promedio Escolar

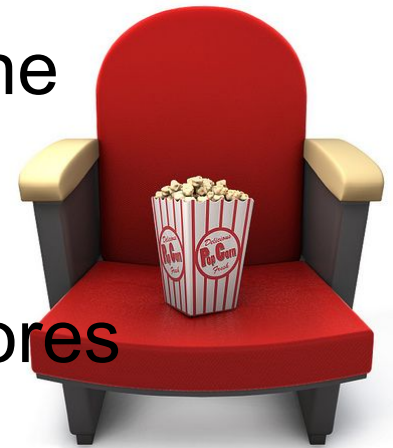
- Desarrolle un algoritmo que permita cargar alumnos y sus notas en los tres trimestres
- Se debe permitir obtener el promedio anual (es decir, de sus tres notas) de un alumno (ingresado por el usuario)
- Luego de resolverlo, pensar en aprovechar métodos y discutir cómo representar la información



Estructuras de Datos y Métodos

Cine

- Diseñar un algoritmo que recorra las butacas de una sala de cine y determine cuántas butacas desocupadas hay
- Suponga que para modelar este problema, se utiliza un arreglo con valores lógicos
 - La presencia de un valor verdadero (true) en el arreglo indica que la butaca está ocupada
 - La presencia de un valor falso (false) en el arreglo indica que la butaca está desocupada



Técnicas de Programación

Programador full-stack

(Resolución)

Estructuras de Datos y Métodos

Clave

Ejercicio: Convertir una palabra ingresada por el usuario en clave, según reglas vistas.

```
import * as rls from "readline-sync";
let palabra : string = rls.question("Indique la palabra a codificar: ");
console.log(`La palabra ingresada: ${palabra} se convierte en: ${convertirEnClave(palabra)}`);

function convertirEnClave(palabra : string) : string {
    let vocales : string = "aeiou";
    let signos : string = ",.;;!";
    let matematicos : string = "0123456789+-*/";
    let clave : string = "";
    for (let index = 0; index < palabra.length; index++) {
        if (matematicos.indexOf(palabra[index]) >= 0) {
            clave += palabra[index];
        } else {
            if (vocales.indexOf(palabra[index]) >= 0) {
                clave += signos[vocales.indexOf(palabra[index])];
            } else {
                if (palabra[index] == palabra[index].toUpperCase())
                    clave += palabra[index].toLowerCase();
                else
                    clave += palabra[index].toUpperCase();
            }
        }
    }
    return clave;
}
```

Estructuras de Datos y Métodos

Camel Case

Ejercicio: Convertir un texto y un nombre de variable/función con las reglas Camel Case

```
import * as rls from "readline-sync";
let texto : string = rls.question("Ingrese un texto: ");
console.log(`El texto: ${texto} convertido a CamelCase queda: ${pasarACamelCase(texto)}`);

function pasarACamelCase(texto : string) : string {
    let aMayusc : boolean = false;
    let camelCase : string = "";
    for (let index = 0; index < texto.trim().length; index++) {
        if (texto[index] == " ") {
            aMayusc = true;
        } else {
            if (aMayusc) {
                camelCase += texto[index].toUpperCase();
                aMayusc = false;
            } else {
                camelCase += texto[index].toLowerCase();
            }
        }
    }
    return camelCase;
}
```

Estructuras de Datos y Métodos

Producto Escalar

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

let cantidad : number = rls.questionInt("Cantidad: ");

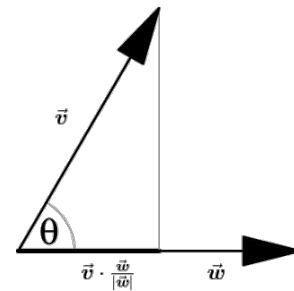
let v1 : number[] = new Array(cantidad);
let v2 : number[] = new Array(cantidad);

console.log("Cargando v1");
cargarVector(v1, cantidad);           //REUSAMOS

console.log("Cargando v2");
cargarVector(v2, cantidad);           //REUSAMOS

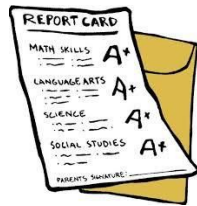
console.log("Multiplicacion escalar = ", multiplicarVector(v1, v2, cantidad) );

function multiplicarVector(v1 : number[], v2 : number[], cantidad : number) : number {
    let acum : number = 0;
    let indice : number;
    for (indice=0; indice<cantidad; indice++) {
        acum = acum + (v1[indice] * v2[indice]);
    }
    return acum;
}
```



Estructuras de Datos y Métodos

Promedio Escolar



```
let numAlum : number = rls.questionInt("Ingrese el número de alumnos: ");
```

```
let alumnos : string[] = new Array(numAlum);
```

```
let nota1 : number[] = new Array(numAlum);
```

```
let nota2 : number[] = new Array(numAlum);
```

```
let nota3 : number[] = new Array(numAlum);
```

```
let indice : number;
```

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

```
    let alumnos[indice] = rls.question("Nombre: ");
```

```
    let nota1[indice] = rls.questionInt("Nota 1er trimestre:");
```

```
    let nota2[indice] = rls.questionInt("Nota 2do trimestre:");
```

```
    let nota3[indice] = rls.questionInt("Nota 3er trimestre:");
```

```
}
```

```
let alumBuscado : string = rls.question("A quien busca: ");
```

```
let encontrado : boolean = false;
```

```
let promedio : number = 0;
```

```
indice = 0;
```

```
while (indice < alumNum && !encontrado) {
```

```
    if (alumnos[indice] == alumBuscado) {
```

```
        encontrado = true;
```

```
        promedio = nota1[indice] + nota2[indice] + nota3[indice];
```

```
        promedio /= 3;
```

```
    }
```

```
    indice++;
```

```
}
```

```
if (encontrado) {
```

```
    console.log("El promedio de ",  
    alumBuscado, " es ", promedio);
```

```
} else {
```

```
    console.log("No se pudo encontrar a ",  
    alumBuscado);
```

```
}
```

Estructuras de Datos y Métodos

Cine



```
function cargarButacas(arreglo : boolean[], largo : number) {  
    let indice : number;  
    for (indice=0; indice<largo; indice++) {  
        arreglo[indice] = Math.floor(Math.random() * 2); //numero booleano al azar 0 o 1  
    }  
}  
  
function contarButacasDesocupadas(arreglo : boolean[], largo : number) : number {  
    let desocupadas : number=0;  
    let indice : number;  
    for (indice=0; indice<largo; indice++) {  
        if (arreglo[indice] == 0) {  
            desocupadas++;  
        }  
    }  
    return desocupadas;  
}  
  
let nroButacas : number = 100;  
let butacas : boolean[] = new Array(nroButacas);  
cargarButacas(butacas, nroButacas);  
console.log("El número de butacas desocupadas es: ", contarButacasDesocupadas(butacas, nroButacas));
```

Técnicas de Programación

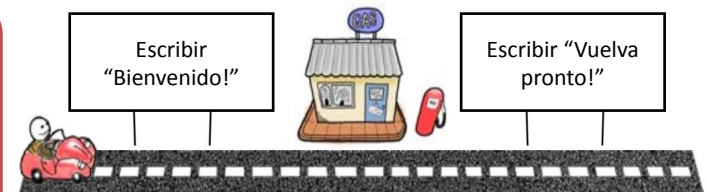
Programador full-stack

Repaso General para el Examen

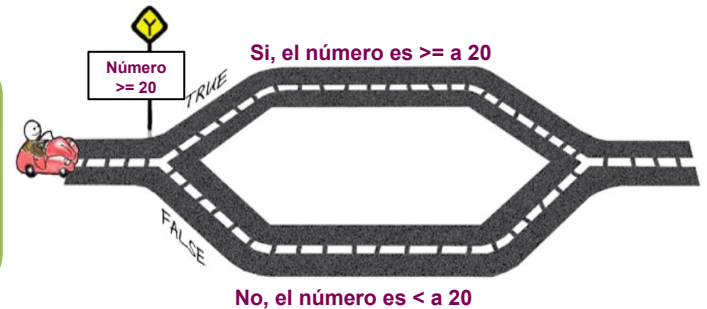
Estructuras de Control

Selección

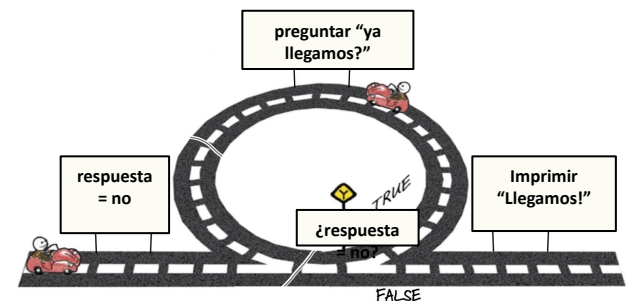
Secuenciales



Selección o de Decisión

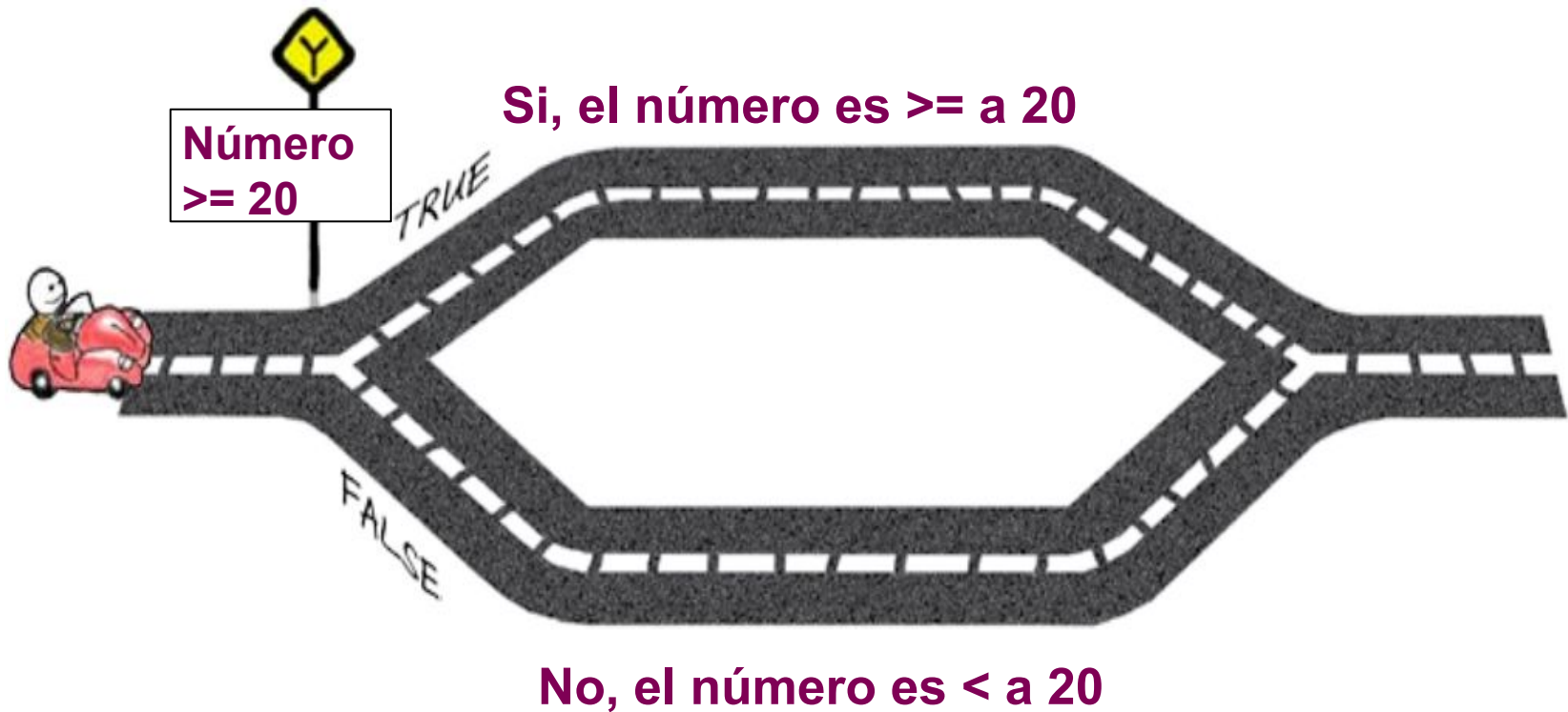


Repetitivas



Estructuras de Control

Selección



Estructura de Control

Selección Simple y Múltiple

Selección Simple

```
if (<condición>) {  
    <instrucciones>  
} else {  
    <instrucciones>  
}
```

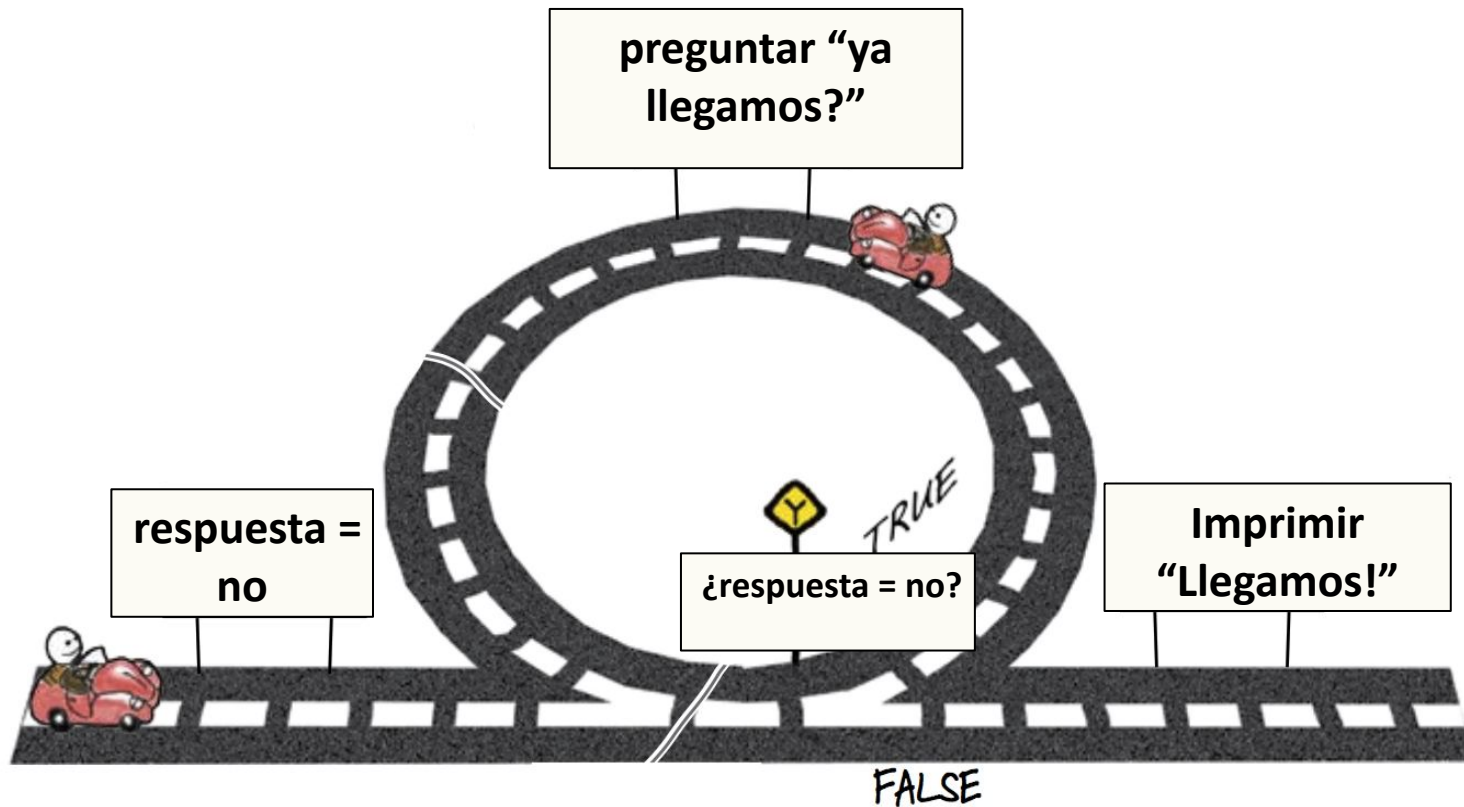


Selección Múltiple

```
switch (<condición>) {  
    case <opción1>: <instrucciones> break;  
    case <opción2>: <instrucciones> break;  
    <...>  
    default: <instrucciones>  
}
```

Estructuras de Control

Iteración / Repetición



Estructuras de Control

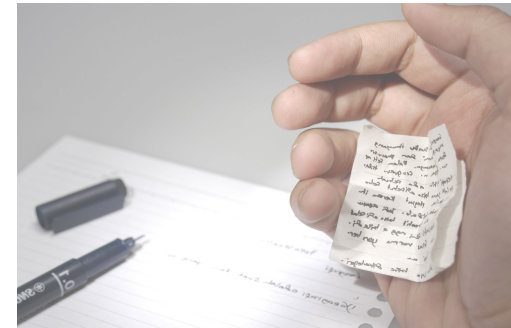
Iteración / Repetición

Repetición
condicional

```
while (<condición>) {  
    <instrucciones>  
}
```

Repetición
acotada

```
for (<varctrl>=<inicial>; <varctrl> < <final>; <varctrl><in/decr>) {  
    <instrucciones>  
}
```



Operadores Condicionales

Operador	Significado	Ejemplo
>	Mayor que	$3 > 1$
<	Menor que	$1 < 3$
==	Igual que	$1 == 1$
>=	Mayor igual que	$4 >= 2$
<=	Menor igual que	$4 <= 2$
!=	Distinto que	$9 != 3$

Operadores Lógicos

Operador	Significado	Descripción	Ejemplo
&&	Conjunción (Y)	Ambas son Verdaderas	$(7 > 4) \ \&\& \ (2 == 2)$
	Disyunción (O)	Al menos una es verdadera	$(1 == 1 \ \ 2 == 1)$
!	Negación (No)	No es verdadero	$!(2 < 5)$

Prueba de Escritorio

- Técnica utilizada para validar la resolución de problemas con algoritmos, de uso frecuente en el ámbito informático
- Sirve para validar utilizando datos reales como ejemplo, un algoritmo definido y así comprobar si se obtiene el resultado deseado
- Ejemplo, recuerde el ejercicio de verificar si un número es mayor a 20. Se podría verificar con un número mayor a 20, un número igual a 20 y un número menor que 20

Estructura de Control - Selección

Mayor a 20 - Prueba de Escritorio

Código	Datos Entrada	Respuesta Deseada
<pre>//Algoritmo Mayor20 let nroDeseado : number; nroDeseado=rls.questionInt("Escriba el número que desea verificar si es mayor o no a 20: "); if (nroDeseado > 20) { console.log('El número es mayor a 20: ',nroDeseado); } else { console.log('El número es menor o igual a 20: ',nroDeseado); }</pre>	nroDeseado = 20	El número es menor o igual a 20: 20
	nroDeseado = 3	El número es menor o igual a 20: 3
	nroDeseado = 45	El número es mayor a 20: 45

Métodos

- **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

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

The diagram illustrates the execution flow of a function call. A blue arrow originates from the `dibujar40Guiones()` call within the `if` block and points to the start of the `function dibujar40Guiones()` definition. Another blue arrow starts from the closing curly brace of the `function` and points back to the line of code immediately following the function call, `console.log("El resultado de la operacion es: ", numero1+numero2);`, demonstrating how the program returns to the point of the call after executing the function's code.

Métodos

Parámetros

- Son valores que enviamos a los métodos
- Se inicializa fuera del método
- Tienen un tipo
- Dentro del método se comporta como una variable
- Nos ayudan a evitar métodos duplicados

```
function dibujar30Guiones() {
  let x : number, linea : string;
  for (x=1; x<=30; x++) {
    linea += "-";
  }
  console.log(linea);
}

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

```
dibujar30Guiones();
dibujar40Guiones();
```

cantidad es un parámetro y nos permite indicar cuantos guiones queremos dibujar

```
function dibujarGuiones(cantidad) {
  let x : number, linea : string;
  for (x=1; x<=cantidad; x++) {
    linea += "-";
  }
  console.log(linea);
}
```

```
dibujarGuiones(30);
dibujarGuiones(40);
dibujarGuiones(20);
```

Métodos

Retorno

- Análogamente se puede utilizar para retornar algun valor

```
function dibujarGuiones(cantidad) : string {  
  let x : number, linea : string;  
  for (x=1; x<=cantidad; x++) {  
    linea += "-";  
  }  
  return linea;  
}
```

la funcion ya no envia a consola los guiones sino que los retorna como texto al programa llamador.

```
console.log(dibujarGuiones(30));
```

el programa llamador, puede enviar el resultado a consola directamente

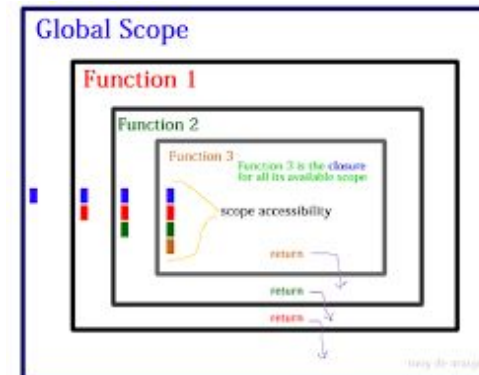
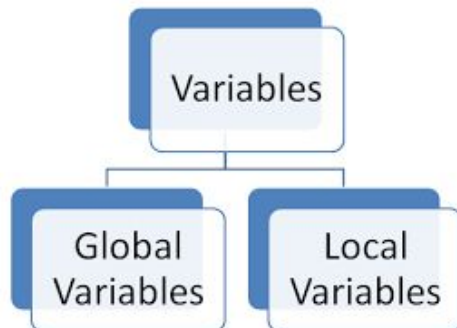
```
let guiones = dibujarGuiones(40);  
console.log(guiones);
```

o bien guardar el resultado en una variable y luego enviarla a consola

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.
 - Indentación en las estructuras de control
 - Comentarios en el código
- 
- *//Comentario de línea en typescript*
 - */*Esto es un comentario de bloque, permite escribir más de una línea */*

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

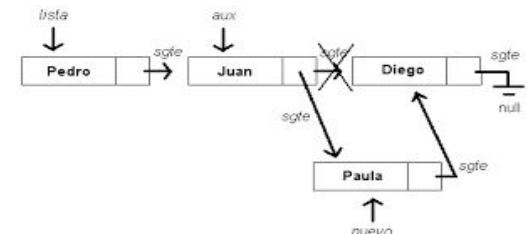
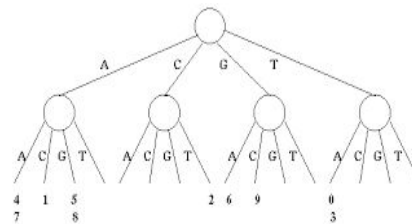


Estructuras de Datos

Forma particular de organizar datos



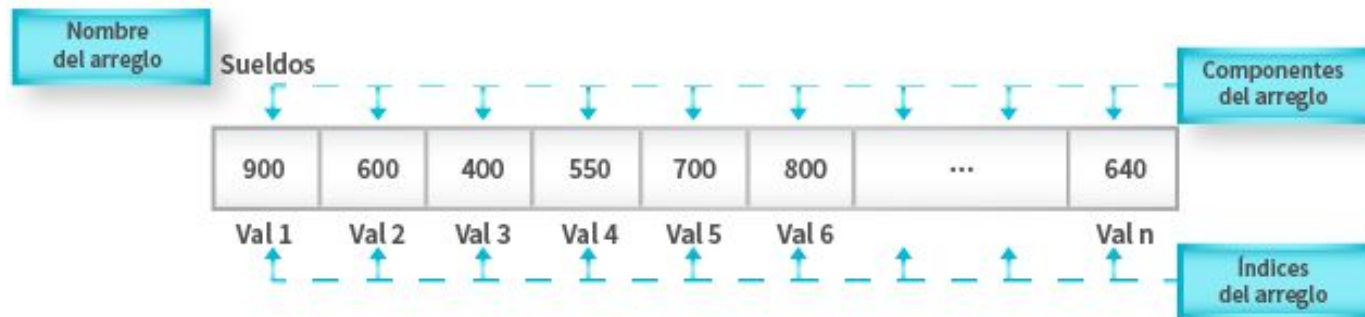
- Estructuras que permiten **COLECCIONAR** elementos
 - GUARDARLOS
 - RECORRERLOS
 - MANIPULARLOS
 - Operaciones básicas
 - **COLOCAR**
 - **OBTENER**
- Diagrama de un árbol de búsqueda binaria para el alfabeto de ADN (A, C, G, T). El árbol tiene una raíz que se divide en cuatro ramas etiquetadas A, C, G y T. Cada rama conduce a un nodo que a su vez se divide en cuatro ramas etiquetadas A, C, G y T. Los nodos de la tercera generación están etiquetados con números: 4, 1, 5, 8; 2, 6, 9; 0, 3.
- Estructuras
 - **LISTAS**
 - **COLAS**
 - **PILAS**
 - **ARBOLES**
- Diagrama de una lista enlazada. Se muestran dos nodos rectangulares. El primer nodo contiene 'Pedro' y tiene una flecha etiquetada 'sigte' que apunta al segundo nodo. El segundo nodo contiene 'Juan' y tiene una flecha etiquetada 'sigte' que apunta a un tercer nodo parcialmente visible. Arriba de los nodos hay etiquetas 'lista' y 'aux' con flechas que apuntan a los nodos.



Estructuras de Datos

Arreglos / Listas / Vectores

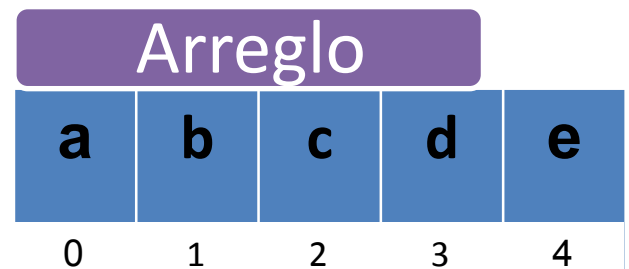
- Los arreglos son estructuras de datos homogéneas (todos sus datos son del mismo tipo)
- Permiten almacenar un determinado número de datos
- Tiene muchos elementos, y a cada uno de ellos se acceden indicando que posición se quiere usar (un índice)



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 – 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 / 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 Números - Código

//Algoritmo ArregloNumeros

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

Definición del arreglo num
con dimensión 7

```
let indice : number;
```

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

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

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

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

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

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

```
num[6] = 4;
```

```
indice = 0;
```

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

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

```
    indice++;
```

```
}
```

Estructuras de Datos – Arreglos

Ejercicio – Arreglo de Números - Código

//Algoritmo ArregloNumeros

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

```
let indice : number;
```

```
num[0] = 20;  
num[1] = 14;  
num[2] = 8;  
num[3] = 0;  
num[4] = 5;  
num[5] = 19;  
num[6] = 4;
```

Se completa el arreglo con
números fijos

```
indice = 0;
```

```
while (indice < 7) {  
    console.log ("El número en la posición ", indice, " es ", num[indice]);  
    indice++;  
}
```

Estructuras de Datos – Arreglos

Ejercicio – Arreglo de Números - Código

//Algoritmo ArregloNumeros

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

```
let indice : number;
```

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

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

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

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

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

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

```
num[6] = 4;
```

```
indice = 0;
```

Se inicializa el índice para comenzar a recorrer el arreglo desde la posición 0

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

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

```
    indice++;
```

```
}
```

Estructuras de Datos – Arreglos

Ejercicio – Arreglo de Números - Código

//Algoritmo ArregloNumeros

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

```
let indice : number;
```

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

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

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

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

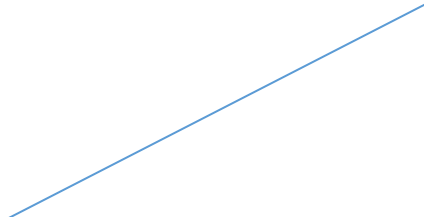
```
num[4] = 5;
```

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

```
num[6] = 4;
```

```
indice = 0;
```

Recorre el arreglo mostrando
los números que posee



```
while (indice < 7) {  
    console.log ("El número en la posición ", indice, " es ", num[indice]);  
    indice++;  
}
```

Estructuras de Datos – Arreglos

Ejercicio – Arreglo de Números - Código

//Algoritmo ArregloNumeros

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

```
let indice : number;
```

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

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

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

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

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

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

```
num[6] = 4;
```

```
indice = 0;
```

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

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

```
    indice++;
```

```
}
```

Estructuras de Datos – Arreglos

Ejercicio – Arreglo de Números - Código

//Algoritmo ArregloNumeros

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

```
let indice : number;
```

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

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

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

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

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

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

```
num[6] = 4;
```

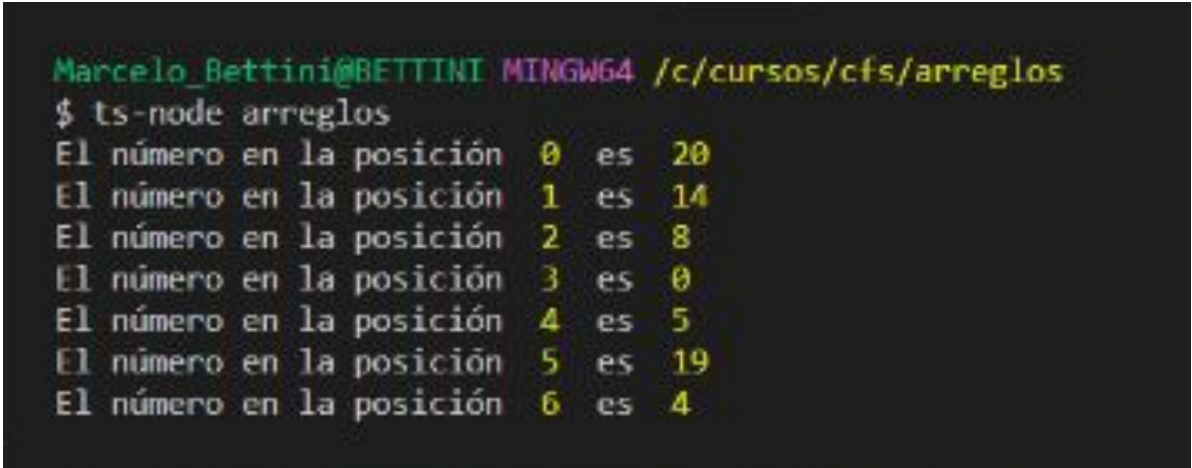
```
indice = 0;
```

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

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

```
    indice++;
```

```
}
```



```
Marcelo_Bettini@BETTINI MINGW64 /c/cursos/cfs/arreglos
$ ts-node arreglos
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 4
```


Estructuras de Datos

Arreglos, Métodos y Pasaje de Parámetros

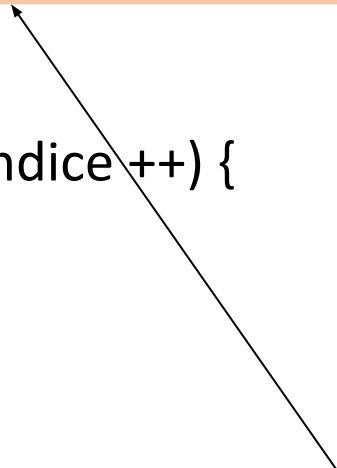
- Podemos **reutilizar** código!
- Las **modificaciones** se pueden hacer **directamente** en los arreglos que pasamos como **parámetro** (solo funciona para arreglos y matrices, no para otros tipos de datos)



Estructuras de Datos y Métodos

Definición con Estructuras como Parámetros

```
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;  
}
```



Al pasar como parámetro un arreglo, también debo indicar su dimensión para poder recorrer todos sus valores

Estructuras de Datos y Métodos

Retornos de Arreglos/Matrices

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

```
  let indice : number;
```

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

```
    v[indice] = rls.questionInt(`Ingrese el valor ${indice} :`);
```

```
  }
```

```
}
```

Las modificaciones se hacen sobre el arreglo declarado como parámetro que es el arreglo original

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)



Programador full-stack

Ejercicios de Repaso

Ejercicios de Repaso

Ejercicio – Calcular Promedio

- El DT de los infantiles del equipo de fútbol desea saber el promedio de la edad de los chicos
- La edad de los chicos va de 3 a 7 años. Las edades son cargadas al azar (use la función `aleatorio(menorValor, mayorValor)`, es decir `aleatorio(3,7)`)
- Muestre todas las edades y el promedio de las mismas


$$\begin{array}{r} 6 + 8 + 9 + 6 + 9 + 7 + 8 + 5 = 58 \\ \text{Icon 1} + \text{Icon 2} + \text{Icon 3} + \text{Icon 4} + \text{Icon 5} + \text{Icon 6} + \text{Icon 7} + \text{Icon 8} = 8 \\ \hline 58 \div 8 = 7.2 \end{array}$$

Ejercicios de Repaso

Ejercicio – Calcular Promedio

$$6 + 8 + 9 + 6 + 9 + 7 + 8 + 5 = 58$$

$$\frac{58}{8} = 7.2$$

```
//Genera un número aleatorio entre min y max
function Aleatorio(min : number, max : number) : number {
    return Math.floor(Math.random() * (max - min + 1) ) + min;
}

//Calcula el promedio de las edades de los jugadores de fútbol
let promedio : number = 0;
let dimArreglo : number = rls.questionInt(`Indique la cantidad de jugadores: `);
let numArreglo : number[] = new Array (dimArreglo);
cargarArreglo(numArreglo, dimArreglo);
mostrarArreglo(numArreglo, dimArreglo);
promedio = obtenerPromedio(numArreglo, dimArreglo);
console.log (`El promedio las edades es de: ${promedio}`);
```

Ejercicios de Repaso

Ejercicio – Calcular Promedio - Errores

$$\begin{array}{c} 6 + 8 + 9 + 6 + 9 + 7 + 8 + 5 = 58 \\ \text{[person icons]} = \frac{58}{8} = 7.2 \end{array}$$

//Completa un arreglo con números aleatorios del 3 al 7

```
function cargarArreglo(numeroArreglo:number[],dimensionArreglo:number) {  
    for ( indice = 0 ; indice < dimensionArreglo; indice++) {  
        numeroArreglo[indice] = Aleatorio(3,7);  
    }  
}
```



Ejercicios de Repaso

Ejercicio – Calcular Promedio - Errores

$$\begin{array}{c}
 6 + 8 + 9 + 6 + 9 + 7 + 8 + 5 = 58 \\
 \text{[icono]} + \text{[icono]} + \text{[icono]} + \text{[icono]} + \text{[icono]} + \text{[icono]} + \text{[icono]} + \text{[icono]} = \frac{58}{8} = 7.2
 \end{array}$$

//Completa un arreglo con números aleatorios del 3 al 7

Falta
definir
indice

```

function cargarArreglo(numeroArreglo:number[],dimensionArreglo:number) {
  for ( indice = 0 ; indice < dimensionArreglo; indice++) {
    numeroArreglo[indimensionArreglo] = Aleatorio(3,7);
  }
}

```

No va
dimensionArreglo
va indice



Ejercicios de Repaso

Ejercicio – Calcular Promedio

$$6 + 8 + 9 + 6 + 9 + 7 + 8 + 5 = 58$$
$$\text{stick figures} = 58$$
$$\frac{58}{8} = 7.2$$

//Completa un arreglo con números aleatorios del 3 al 7

```
function cargarArreglo(numeroArreglo:number[],dimensionArreglo:number) {  
    let indice : number;  
  
    for ( indice = 0 ; indice < dimensionArreglo; indice++) {  
        numeroArreglo[indice] = Aleatorio(3,7);  
    }  
}
```

Ejercicios de Repaso

Ejercicio – Calcular Promedio - Errores

$$\begin{array}{cccccccccccc} 6 & + & 8 & + & 9 & + & 6 & + & 9 & + & 7 & + & 8 & + & 5 & = & 58 \\ \text{child} & + & \text{child} & + & \text{child} & + & \text{child} & + & \text{child} & + & \text{child} & + & \text{child} & + & \text{child} & = & 8 \end{array} = 7.2$$

//Muestra un arreglo

```
function mostrarArreglo(numeroArreglo:number[],dimensionArreglo:number) {  
    let numeroArreglo;  
    for ( indice = 0 ; indice < dimensionArreglo ; indice++) {  
        console.log ( " ", numeroArreglo[indice] );  
    }  
    console.log ( "\n " );  
}
```



Ejercicios de Repaso

Ejercicio – Calcular Promedio - Errores

$$\begin{array}{c}
 6 + 8 + 9 + 6 + 9 + 7 + 8 + 5 = 58 \\
 \text{[person icon]} + \text{[person icon]} + \text{[person icon]} + \text{[person icon]} + \text{[person icon]} + \text{[person icon]} + \text{[person icon]} + \text{[person icon]} = \frac{58}{8} = 7.2
 \end{array}$$

//Muestra un arreglo

```
function mostrarArreglo(numeroArreglo:number[],dimensionArreglo:number) {
```

```
    let numeroArreglo;
```

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

```
        console.log ( " ", numeroArreglo[indice] );
```

```
    }
```

```
    console.log ("\n ");
```

```
}
```



Falta definir
indice

Los parametros
no se definen

Ejercicios de Repaso

Ejercicio – Calcular Promedio

$$\begin{array}{l} 6 + 8 + 9 + 6 + 9 + 7 + 8 + 5 = 58 \\ \text{Figure 1} + \text{Figure 2} + \text{Figure 3} + \text{Figure 4} + \text{Figure 5} + \text{Figure 6} + \text{Figure 7} + \text{Figure 8} = 8 \\ \hline 58 \div 8 = 7.2 \end{array}$$

//Muestra un arreglo

```
function mostrarArreglo(numeroArreglo:number[],dimensionArreglo:number) {  
    let indice : number;  
    let linea : string = "";  
    for ( indice = 0 ; indice < dimensionArreglo ; indice++) {  
        linea += ` ${numeroArreglo[indice]} `;  
    }  
    console.log (linea);  
}
```

Ejercicios de Repaso

Ejercicio – Calcular Promedio

$$6 + 8 + 9 + 6 + 9 + 7 + 8 + 5 = 58$$

$$\text{8 figures} = \frac{58}{8} = 7.2$$

//Completa un arreglo con números aleatorios del 3 al 7

```
function cargarArreglo(numeroArreglo:number[],dimensionArreglo:number) {
    let indice : number;
    for (indice = 0 ; indice < dimensionArreglo; indice++) {
        numeroArreglo[indice] = Aleatorio(3,7);
    }
}
```

//Muestra un arreglo

```
function mostrarArreglo(numeroArreglo:number[],dimensionArreglo:number) {
    let linea : string = "";
    let indice : number;
    for (indice = 0 ; indice < dimensionArreglo; indice++) {
        linea += ` ${numeroArreglo[indice]}`;
    }
    console.log (linea);
}
```

Ejercicios de Repaso

Ejercicio – Calcular Promedio - Errores

$$\begin{array}{r}
 6 + 8 + 9 + 6 + 9 + 7 + 8 + 5 = 58 \\
 \text{[person icons]} = \frac{58}{8} = 7.2
 \end{array}$$

// Calcula el promedio de un arreglo

```

function obtenerPromedio(numArreglo:number[],dimArreglo:number) : number {
    let prome:number = 0;
    let sumaTotal:number = 0;
    let indice:number;
    for (indice=0; indice < dimArreglo; indice++) {
        sumaTotal = sumaTotal+numArreglo[indice];
    }
    prome=sumaTotal/dimArreglo;
    return Math.floor(prome);
}
  
```



Ejercicios de Repaso

Ejercicio – Calcular Promedio - Errores

$$\begin{array}{c}
 6 + 8 + 9 + 6 + 9 + 7 + 8 + 5 = 58 \\
 \text{[icono]} + \text{[icono]} + \text{[icono]} + \text{[icono]} + \text{[icono]} + \text{[icono]} + \text{[icono]} + \text{[icono]} = 8
 \end{array}
 = \frac{58}{8} = 7.2$$

// Calcula el promedio de un arreglo

```

function obtenerPromedio(numArreglo:number[],dimArreglo:number) : number {
    let prome:number = 0;
    let sumaTotal:number = 0;
    let indice:number;
    for (indice=0; indice < dimArreglo; indice++) {
        sumaTotal = sumaTotal + numArreglo[dimArreglo];
    }
    prome=sumaTotal/dimArreglo;
    return Math.floor(prome);
}
  
```



No va
dimArreglo
va indice

Ejercicios de Repaso

Ejercicio – Calcular Promedio

$$\begin{array}{l} 6 + 8 + 9 + 6 + 9 + 7 + 8 + 5 = 58 \\ \text{Figure 1} + \text{Figure 2} + \text{Figure 3} + \text{Figure 4} + \text{Figure 5} + \text{Figure 6} + \text{Figure 7} + \text{Figure 8} = \frac{58}{8} = 7.2 \end{array}$$

// Calcula el promedio de un arreglo

```
function obtenerPromedio(numArreglo:number[],dimArreglo:number) : number {  
    let prome:number = 0;  
    let sumaTotal:number = 0;  
    let indice:number;  
    for (indice=0; indice < dimArreglo; indice++) {  
        sumaTotal = sumaTotal + numArreglo[indice];  
    }  
    prome=sumaTotal/dimArreglo;  
    return Math.floor(prome);  
}
```

Ejercicios de Repaso

Ejercicio – Calcular Promedio

$$6 + 8 + 9 + 6 + 9 + 7 + 8 + 5 = 58$$

$$\frac{58}{8} = 7.2$$

//Calcula el promedio de las edades de los jugadores de fútbol

let promedio : **number** = 0;

let dimArreglo : **number** = rls.questionInt(`Indique la cantidad de jugadores: `);

let numArreglo : **number**[] = **new Array** (dimArreglo);

cargarArreglo(numArreglo, dimArreglo);

mostrarArreglo(numArreglo, dimArreglo);

promedio = obtenerPromedio(numArreglo, dimArreglo);

console.log (`El promedio las edades es de: \${promedio}`);

```
Marcelo_Bettini@BETTINI MINGW64 /c/cursos/cfs/ejercicio
$ ts-node mediaCalc
Indique la cantidad de jugadores: 8
  45  17  45  75  50  64  82  68
El promedio las edades es de: 55
```

Ejercicios de Repaso

Ejercicio – Sumar Dos Arreglos

- Hacer la suma de dos arreglos y dejarlo en otro arreglo
- La dimensión de los arreglos es solicitada al usuario
- Los dos arreglos son cargados al azar

Ejercicios de Repaso

Ejercicio – Sumar Dos Arreglos

```
// Calcula la suma de tres arreglos
let dim : number = rls.questionInt(`Ingrese la dimensión del arreglo: `);
let arreglo1 : number[] = new Array (dim);
let arreglo2 : number[] = new Array (dim);
let arreglo3 : number[] = new Array (dim);
cargarArreglo(arreglo1, dim);
cargarArreglo(arreglo2, dim);
sumarArreglos(arreglo1, arreglo2, arreglo3, dim);
mostrarArreglo(arreglo1, dim);
mostrarArreglo(arreglo2, dim);
console.log ("La suma de los arreglos es:")
mostrarArreglo(arreglo3, dim);
```

Ejercicios de Repaso

Ejercicio – Sumar Dos Arreglos - Errores



//Completa un arreglo con números al azar menores que 100

```
function cargarArreglo(arreglo:number[], dim:number) {  
    for ( indice = 0 ; indice < dim; indice++) {  
        arreglo[indice] = Math.random(100);  
    }  
}
```

//Suma dos arreglos y el resultado lo pone en otro arreglo

```
function sumarArreglos(arreglo1:number[], arreglo2:number[], arreglo3:number[], dim:number)  
    let indice;  
    for ( indice = 0 ; indice <= dim; indice++) {  
        arreglo3[indice] = arreglo1[indice] + arreglo2[indice];  
    }  
}
```

Ejercicios de Repaso

Ejercicio – Sumar Dos Arreglos - Errores

No define
indice

No es dim
sino indice



//Completa un arreglo con números al azar menores que 100

```
function cargarArreglo(arreglo:number[], dim:number) {  
  for ( indice = 0 ; indice < dim; indice++) {  
    arreglo[dim] = Math.floor(Math.random()*100);  
  }  
}
```

//Suma dos arreglos y el resultado lo pone en otro arreglo

```
function sumarArreglos(arreglo1:number[], arreglo2:number[], arreglo3:number[], dim:number)  
  let indice:number;  
  for ( indice = 0 ; indice <= dim; indice++) {  
    arreglo3[indice] = arreglo1[dim] + arreglo3[dim];  
  }  
}
```

Es <

No es arreglo3
sino arreglo2

Ejercicios de Repaso

Ejercicio – Sumar Dos Arreglos

//Completa un arreglo con números al azar menores que 100

```
function cargarArreglo(arreglo:number[], dim:number) {  
    let indice:number;  
    for ( indice = 0 ; indice < dim; indice++) {  
        arreglo[indice] = Math.floor(Math.random()*100);  
    }  
}
```

//Suma dos arreglos y el resultado lo pone en otro arreglo

```
function sumarArreglos(arreglo1:number[], arreglo2:number[], arreglo3:number[], dim:number) {  
    let indice:number;  
    for ( indice = 0 ; indice < dim; indice++) {  
        arreglo3[indice] = arreglo1[indice] + arreglo2[indice];  
    }  
}
```

Ejercicios de Repaso

Ejercicio – Sumar Dos Arreglos

```
//Muestra un arreglo
function mostrarArreglo(arreglo:number[], dimension:number) {
    let linea : string = "";
    let indice : number;
    for (indice = 0 ; indice < dimension; indice++) {
        linea += ` ${arreglo[indice]}`;
    }
    console.log (linea);
}
```


Ejercicios de Repaso

Ejercicio – Sumar Dos Arreglos

```
// Calcula la suma de tres arreglos
let dim : number = rls.questionInt(`Ingrese la dimensión del arreglo: `);
let arreglo1 : number[] = new Array (dim);
let arreglo2 : number[] = new Array (dim);
let arreglo3 : number[] = new Array (dim);
cargarArreglo(arreglo1, dim);
cargarArreglo(arreglo2, dim);
sumarArreglos(arreglo1, arreglo2, arreglo3, dim);
mostrarArreglo(arreglo1, dim);
mostrarArreglo(arreglo2, dim);
console.log ("La suma de los arreglos es:")
mostrarArreglo(arreglo3, dim);
```

```
Marcelo_Bettini@BETTINI MINGW64 /c/cursos/cfs/ejercicio
$ ts-node twoArrSum
Ingrese la dimensión del arreglo: 8
 12 85 4 68 83 5 58 32
 52 57 2 28 18 7 63 4
La suma de los arreglos es:
 64 142 6 96 101 12 121 36
```

Ejercicios de Repaso

Ejercicio – Personas en una Disco

- Para tener control de la gente que hay en una disco el gerente quiere saber cuantas personas de diferentes edades han entrado.
- No se han permitido la entrada a menores de 18 ni mayores de 40. Para la carga de los datos se usa la función aleatorio (use la función aleatorio(menorValor,mayorValor), es decir aleatorio(19,40))
- Se sabe que la cantidad total de personas dentro del local es de 270
- Se quiere saber:
 - Cuántas personas son menores de 21 años
 - Cuántas personas mayores o igual a 21 años
 - Cuántas personas en total



Ejercicios de Repaso

Ejercicio – Personas en una Disco

// Indica la cantidad de menores y de mayores de 21 años que hay en la disco

```
let capacidad = 270;
```

```
let personas : number[] = new Array(capacidad );
```

```
let menores21 = 0;
```

```
let mayores21 = 0;
```

```
completarBoliche(personas, capacidad)
```

```
menores21 = contarMenores(personas, capacidad)
```

```
mostrarPersonas(personas, capacidad)
```

```
console.log ("Los menores de 21 son: ", menores21);
```

```
console.log ("Los mayores de 21 son: ", capacidad - menores21 );
```

```
console.log ("En total hay ", capacidad, " personas");
```



Ejercicios de Repaso

Ejercicio – Personas en una Disco - Errores

// Completa un arreglo con números enteros ingresados al azar

```
function completarBoliche(personas : number[], capacidad : number) {  
    let indice : number;  
  
    for (capacidad=0; capacidad < indice; capacidad++) {  
        personas[indice]=Aleatorio(18,40) ;  
    }  
}
```



Ejercicios de Repaso

Ejercicio – Personas en una Disco - Errores

// Completa un arreglo con números enteros ingresados al azar

function completarBoliche(personas : **number**[], capacidad : **number**) {

let indice : **number**;

for (**capacidad=0**; capacidad < **indice**; capacidad++) {

 personas[indice]=Aleatorio(18,40);

 }

}



No es indice,
es capacidad

No es capacidad,
es indice



Ejercicios de Repaso

Ejercicio – Personas en una Disco

// Completa un arreglo con números enteros ingresados al azar

```
function completarBoliche(personas : number[], capacidad : number) {  
    let indice : number;  
  
    for (indice = 0; indice < capacidad; indice++) {  
        personas[indice]=Aleatorio(18,40) ;  
    }  
}
```



Ejercicios de Repaso

Ejercicio – Personas en una Disco - Errores

// Cuenta la cantidad de menores de 21 que hay en un arreglo

```
function contarMenores(personas : number[], capacidad : number) {  
    let indice : number;  
    for (indice=0; indice < capacidad; indice++) {  
        if (personas[indice] > 12) {  
            menores++;  
        }  
    }  
    return menores;  
}
```



Ejercicios de Repaso

Ejercicio – Personas en una Disco - Errores

// Cuenta la cantidad de menores de 21 que hay en un arreglo

```
function contarMenores(personas : number[], capacidad : number) {
```

```
    let indice : number;
```

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

```
        if (personas[menores] > 12) {
```

```
            menores++;
```

```
        }
```

```
    }
```

```
    return menores;
```

```
}
```

No es > 12,
es < 21

No es menores,
es indice

No inicializa
menores

No define
menores



Ejercicios de Repaso

Ejercicio – Personas en una Disco

// Completa un arreglo con números enteros ingresados al azar

```
function completarBoliche(personas : number[], capacidad : number) {  
    let indice : number;  
    for (indice=0; indice < capacidad; indice++) {  
        personas[indice]=Aleatorio(18,40) ;  
    }  
}
```

// Cuenta la cantidad de menores de 21 que hay en un arreglo

```
function contarMenores(personas : number[], capacidad : number) : number {  
    let menores : number = 0;  
    let indice : number;  
    for (indice=0; indice < capacidad; indice++) {  
        if (personas[indice] > 21) {  
            menores++;  
        }  
    }  
    return menores;  
}
```



Ejercicios de Repaso

Ejercicio – Personas en una Disco

```
// Muestra las edades de cada persona
function mostrarPersonas(personas : number[], capacidad : number) {
  let indice : number;
  let lista : string = "";
  for (indice=1; indice <= capacidad; indice++) {
    lista += ` ${personas[indice-1]}`;
    if (indice % 30 == 0) {
      console.log (lista);
      lista = "";
    }
  }
  console.log (lista);
}
```



Ejercicios de Repaso

Ejercicio – Personas en una Disco

// Indica la cantidad de menores y de mayores de 21 años que hay en la disco

const capacidad = 270;

let personas : **number**[] = **new Array**(capacidad);

let menores21 = 0;

let mayores21 = 0;

completarBoliche(personas, capacidad)

menores21 = contarMenores(personas, capacidad)

mostrarPersonas(personas, capacidad)

console.log ("Los menores de 21 son: ", menores21);

console.log ("Los mayores de 21 son: ", capacidad - menores21);

console.log ("En total hay ", capacidad, " personas");



```

Marcelo_Bettini@BETTINI MINGW64 /c/cursos/cfs/ejercicio
$ ts-node peopleAtDisco
38 36 24 36 39 36 21 34 21 35 23 24 40 39 21 40 31 21 36 28 32 22 30 25 20 20 19 18 26
24
37 34 29 33 29 30 37 27 33 29 31 22 35 34 18 20 34 22 38 36 33 23 27 29 39 33 29 38 19
36
35 18 22 31 38 34 23 26 25 39 31 23 34 23 21 22 25 28 22 18 39 21 26 25 21 19 27 33 22
33
34 36 23 33 26 28 37 36 38 28 29 39 21 27 33 40 35 38 21 33 40 27 24 19 32 37 26 37 21
34
18 39 32 25 27 32 22 36 30 29 19 38 18 25 38 36 23 20 38 19 23 25 23 27 28 36 18 40 20
21
25 19 33 23 32 23 18 30 29 39 27 31 31 30 39 30 31 36 25 29 23 23 34 36 30 38 25 30 26
38
36 22 24 35 33 38 34 23 23 31 37 18 36 30 39 35 20 25 25 32 40 40 25 32 28 21 39 31 34
29
37 26 38 34 40 18 22 24 40 40 22 38 34 24 30 30 38 23 26 31 20 34 23 21 25 31 35 29 39
18
36 30 21 25 38 26 33 33 29 30 24 23 19 23 19 36 23 30 23 29 31 29 38 29 37 29 36 36 24
18

Los menores de 21 son: 228
Los mayores de 21 son: 42
En total hay 270 personas
  
```

Repaso Examen

Ejercicio – Convertir texto en CamelCase

Escribir un algoritmo que pida al usuario ingresar una serie de textos, hasta que ingrese un texto vacío, y los convierta en nombres de variable que cumpla las recomendaciones de buenas prácticas.

Por ejemplo si el usuario ingresa: **cantidad de pasos** lo debe convertir en **cantidadDePasos**

Repaso Examen

Ejercicio – Convertir texto en CamelCase

// Recibe un texto y lo convierte en un nombre de variable estilo CamelCase

```
function convertirACamelCase(texto : string) : string {  
    let textoLocal : string = texto.toLowerCase().trim();  
    let vbleCamelCase : string = '';  
    let indice : number = 0;  
    while (indice < textoLocal.length) {  
        if (textoLocal[indice] == ' ') {  
            vbleCamelCase += textoLocal[indice+1].toUpperCase();  
            indice+=2;  
        } else {  
            vbleCamelCase += textoLocal[indice];  
            indice++;  
        }  
    }  
    return vbleCamelCase;  
}
```

Repaso Examen

Ejercicio – Convertir texto en CamelCase

```
let texto : string = rls.question('Ingrese el texto a convertir a CamelCase : ');
while (texto != "") {
    console.log(convertirACamelCase(texto));
    texto = rls.question('Ingrese el texto a convertir a CamelCase : ');
}
```

```
Usuario@DESKTOP-Q9KCHLH MINGW64 /c/cursos/cfs/ejercicios
$ ts-node camelcase
Ingrese el texto a convertir a CamelCase : este texto se muestra en camel case
esteTextoSeMuestraEnCamelCase
Ingrese el texto a convertir a CamelCase :

Usuario@DESKTOP-Q9KCHLH MINGW64 /c/cursos/cfs/ejercicios
$
```

Repaso Examen

Ejercicio – Decir Que Hace!

```
function metodo1(arr : number[], x : number, y : number) {  
  let ax: number;  
  ax = arr[x];  
  arr[x] = arr[y];  
  arr[y] = ax;  
}
```



Repaso Examen

Ejercicio – Decir Que Hace!

```
function metodo1(arr : number[], x : number, y : number) {  
    let ax: number;  
    ax = arr[x];  
    arr[x] = arr[y];  
    arr[y] = ax;  
}
```

Este método permite intercambiar los valores en las posiciones “**x**” e “**y**” de un arreglo “**arr**” utilizando una variable auxiliar “**ax**”



Repaso Examen

Ejercicio – Decir Que Hace!

```
function metodo2(v : number[], s : number) {  
  let i:number, d:number;  
  i = 0;  
  d = s - 1;  
  while (i < d) {  
    metodo1(v, i, d);  
    i = i + 1;  
    d = d - 1;  
  }  
}
```



Repaso Examen

Ejercicio – Decir Que Hace!

```
function metodo2(v : number[], s : number) {  
  let i:number, d:number;  
  i = 0;  
  d = s - 1;  
  while (i < d) {  
    metodo1(v, i, d);  
    i = i + 1;  
    d = d - 1;  
  }  
}
```

- Este método invierte los elementos del arreglo “v” de tamaño “s”
- El arreglo se navega con dos índices, denominados “i” y “d”, los cuales permiten analizar el extremo izquierdo y derecho al mismo tiempo
- El índice “i” es incrementado y “d” es decrementado en cada iteración



Repaso Examen

Consejos finales

- Leer con mucha atención el enunciado.
- Dividir el problema en sub-problemas en la medida de lo posible.
- Declarar e inicializar todas las variables antes de usarlas.
- Agregar comentarios explicando la funcionalidad.
- Poner extrema atención en el uso de condiciones.
- Revisar y ejecutar el código con varios casos de prueba antes de entregarlo.

Buena suerte!!!!