

Técnicas de Programación

Carrera Programador full-stack

Recursividad (Conceptos)

Recursión

- Permite que un método se invoque a sí mismo para realizar una determinada tarea
- Siempre hay una condición que debe cortar la recursión



Recursión

- **Ventajas**

- Soluciona problemas recurrentes
- Permite solucionar problemas complejos con pocas líneas de código

- **Desventajas**

- Puede ser difícil de entender el código
- Produce excesivas demandas de memoria o tiempo de ejecución



Recursión

Imprimir Contenido de un Arreglo

- Sabemos recorrer un arreglo de forma secuencial
- ¿Pero cómo lo hacemos de forma recursiva?

```
function imprimirArregloRec(arreglo:number[], indice:number, largo:number):number {  
  if (indice<=largo) {  
    console.log(`posición ${indice} tiene: ${imprimirArregloRec(arreglo, indice+1, largo)}`);  
  }  
  return arreglo[indice-1];  
}
```



Recursión

Imprimir Contenido de un Arreglo

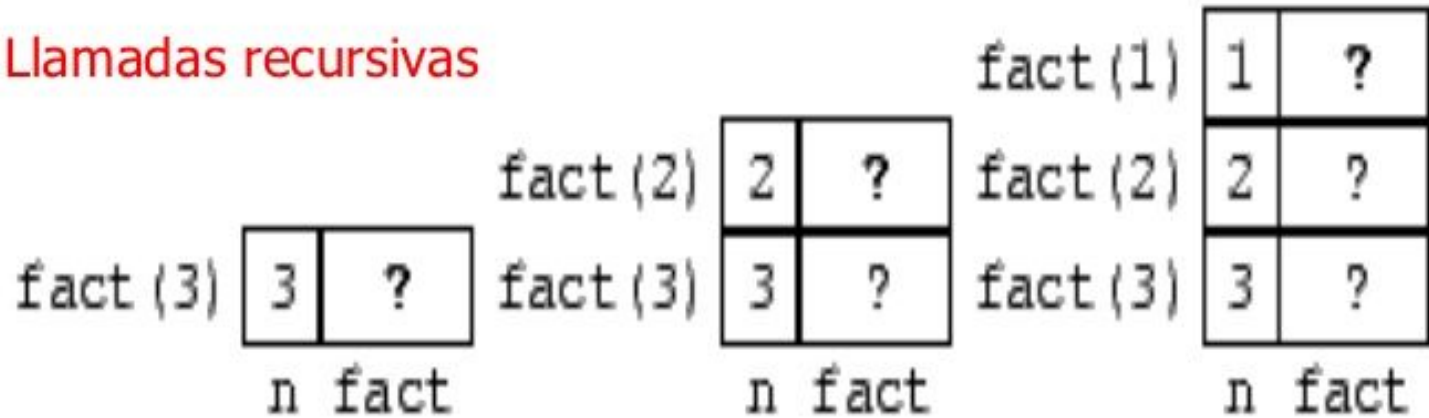
- Sabemos recorrer un arreglo de forma secuencial???

```
function imprimirArregloSec(arreglo:number[],largo:number) {  
  let indice:number;  
  for (indice=0; indice<=largo; indice++) {  
    console.log(`posición ${indice} tiene: ${arreglo[indice]}`);  
  }  
}
```

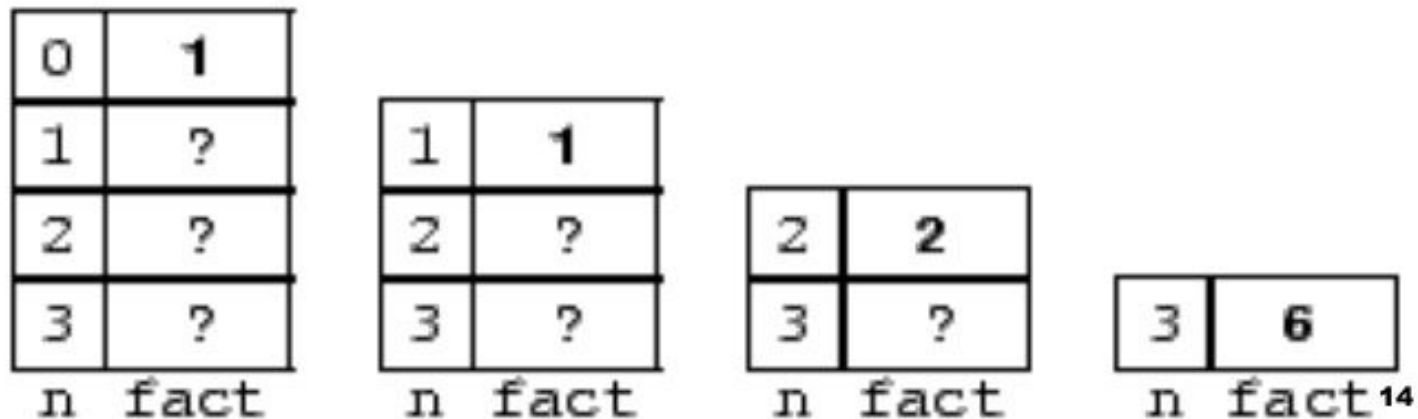
Recursión

Cómo Funciona la Recursividad?

Llamadas recursivas



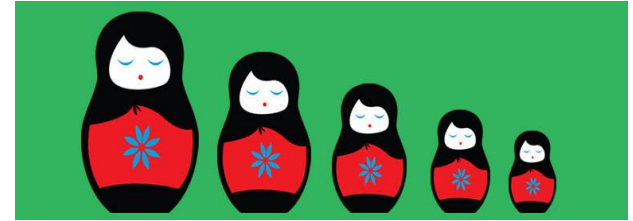
Resultados de las llamadas recursivas



Recursión

Factorial

```
function calcularFactorialRec(n:number):number {  
  let resultado:number = 1;  
  if (n == 0) {  
    resultado = 1;  
  } else {  
    resultado = n * calcularFactorialRec(n-1);  
  };  
  return resultado;  
}
```



```
function calcularFactorialSec(n:number):number{  
  let resultado:number = 1;  
  let indice:number = 1;  
  for(indice = 2;indice <= n; indice++) {  
    resultado = resultado * indice;  
  };  
  return resultado;  
}
```

Técnicas de Programación

Carrera Programador full-stack

Recursión (Ejercicios)

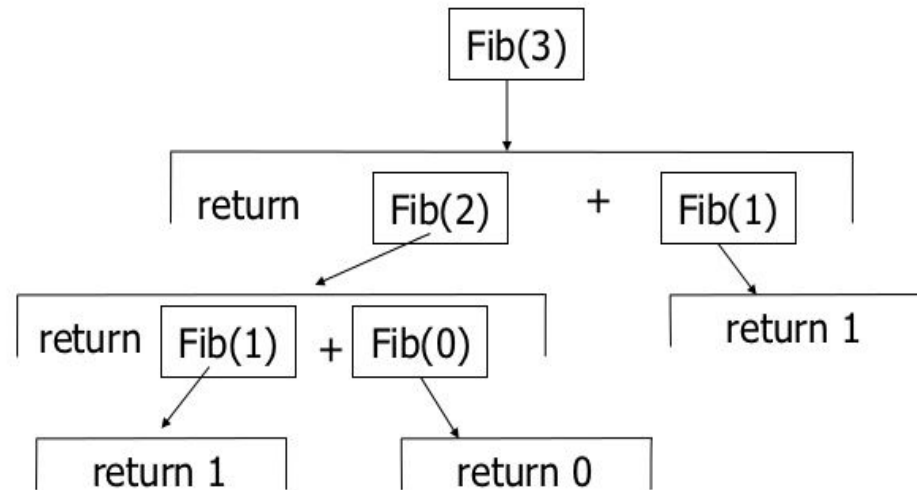
Recursión

Fibonacci

- Fibonacci es una serie numérica
- El Fibonacci de un número n se calcula como:
 - $\text{Fibonacci}(n) = \text{Fibonacci}(n-1) + \text{Fibonacci}(n-2)$
- Excepto:
 - $\text{Fibonacci}(0) = 0$
 - $\text{Fibonacci}(1) = 1$

¿cómo lo
implementamos de
forma recursiva?

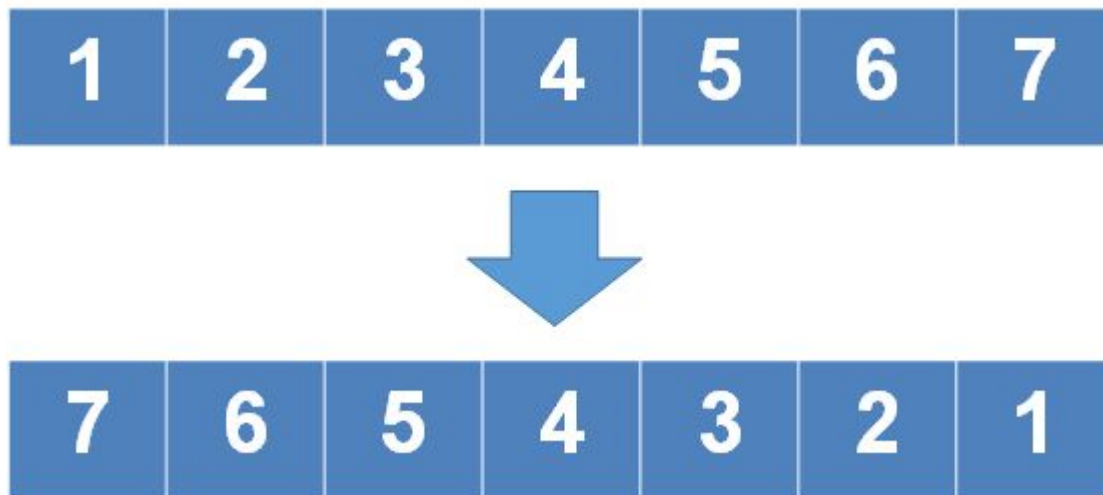
¿Cómo lo
implementamos de
forma secuencial?



Recursión

Invertir Arreglos

- Anteriormente invertimos arreglos de forma secuencial
- ¿Cómo podemos hacerlo de forma recursiva?



Técnicas de Programación

Carrera Programador full-stack

Recursión (Resolución)

Recursión

Fibonacci

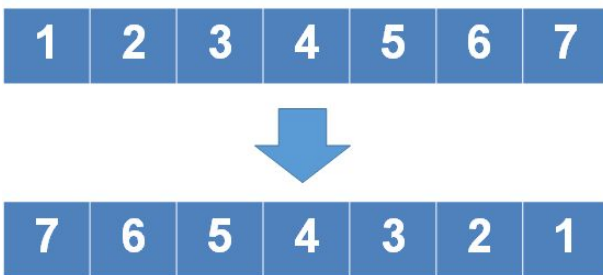
```
function calcularFibonacciRec(n:number):number {  
  let resultado:number = 0;  
  if (n>1) {  
    resultado = calcularFibonacciRec(n-1)+calcularFibonacciRec(n-2);  
  } else {  
    resultado = n;  
  };  
  return resultado;  
}
```

```
function calcularFibonacciSec(n:number):number {  
  let resultado:number = 0;  
  let indice:number, aux1:number, aux2:number;  
  aux1 = 1;  
  for (indice=1; indice <=n; indice++) {  
    aux2=resultado;  
    resultado=aux1;  
    aux1=aux2+aux1;  
  };  
  return resultado;  
}
```

Recursión

Invertir Arreglos

```
function invertirArregloRec(arreglo:number[], indicelzq:number, indiceDer:number) {  
  let aux:number;  
  if (indichelzq<indiceDer) {  
    aux=arreglo[indichelzq];  
    arreglo[indichelzq]=arreglo[indiceDer];  
    arreglo[indiceDer]=aux;  
    invertirArregloRec(arreglo, indicelzq+1, indiceDer-1);  
  };  
}
```



```
function invertirArregloSec(arreglo:number[], largo:number) {  
  let aux:number;  
  let indice:number;  
  for (indice=0; indice < largo - indice; indice++) {  
    aux=arreglo[indice];  
    arreglo[indice]=arreglo[largo-indice];  
    arreglo[largo-indice]=aux;  
  }  
}
```

Recursión

Invertir Arreglos funciones necesarias

```
function imprimirArreglo(arreglo:number[],largo:number) {  
  let indice:number;  
  
  for (indice=0; indice<=largo; indice++) {  
    console.log(`posición ${indice} tiene: ${arreglo[indice]}`);  
  }  
}
```

Recursión

Invertir Arreglos funciones necesarias

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

let n : number = rls.questionInt("cantidad de elementos del arreglo: ");
let arreglo : number[] = new Array(n);
let indice : number;
for (indice=0; indice<=n; indice++) {
    arreglo[indice] = indice*5;
};
imprimirArreglo(arreglo, n);
invertirArregloRec(arreglo, 0, n);
console.log("-----ARREGLO INVERTIDO-----");
imprimirArreglo(arreglo, n);
console.log("-----");
invertirArregloSec(arreglo, n);
console.log("-----VOLVER A INVERTIR-----");
imprimirArreglo(arreglo, n);
```

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.

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

m Columnas

n Filas

Matrices

Para recorrer una matriz necesitamos 2 índices

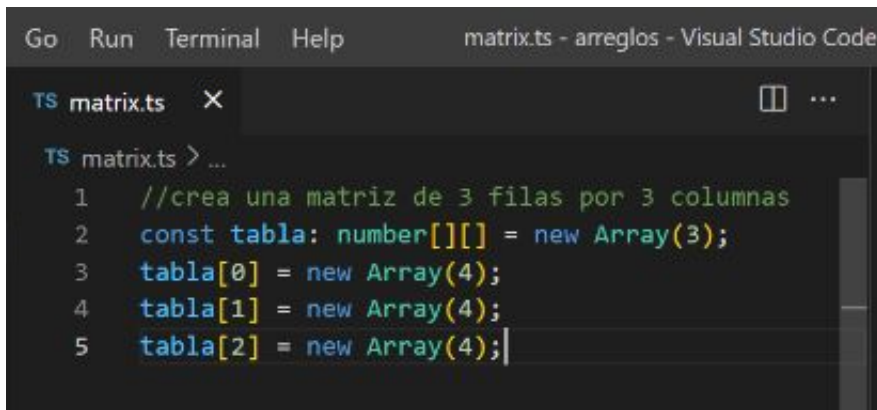
```
let fila : number, columna : number;  
for (columna = 0 ; columna < nroColumnas ; columna++) {  
    for (fila = 0 ; fila < nroFilas ; fila++) {  
        console.log (matriz[fila][columna], " ");  
    }  
}
```

Matrices

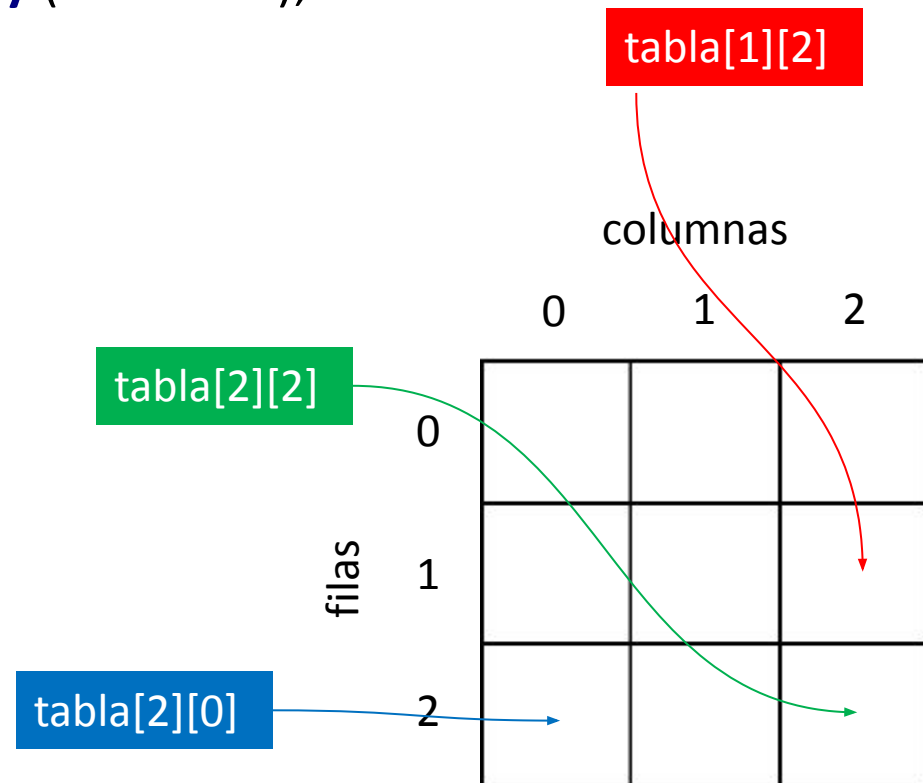
```
identificador : tipo[][] = new Array (filas);  
identificador[0] : tipo[] = new Array (columnas);  
identificador[1] : tipo[] = new Array (columnas);  
...  
identificador[filas-1] : tipo[] = new Array (columnas);
```

- *Ejemplo:* tabla (3,3)

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



```
Go Run Terminal Help matrix.ts - arreglos - Visual Studio Code  
TS matrix.ts x  
TS matrix.ts > ...  
1 //crea una matriz de 3 filas por 3 columnas  
2 const tabla: number[][] = new Array(3);  
3 tabla[0] = new Array(4);  
4 tabla[1] = new Array(4);  
5 tabla[2] = new Array(4);
```



Matrices

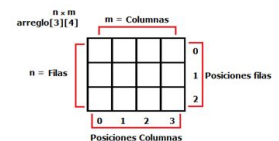
Completa una matriz de $f \times c$ con valores aleatorios entre 0 y “numAzar”

```
function cargar(matriz:number[][], f:number, c:number, numAzar:number) {  
  let fil:number, col:number;  
  for (fil = 0; fil < f; fil++) {  
    for (col = 0; col < c; col++) {  
      matriz[fil][col] = Azar(numAzar);  
    }  
  }  
}
```

Matrices

Muestra por pantalla una matriz de fxc de a una fila por línea

```
function mostrarMatriz(matriz:number[][], f:number, c:number) {
  let fil:number, col:number;
  let cadena:string;
  for (fil = 0; fil < f; fil++) {
    cadena = " ";
    for (col = 0; col < c; col++) {
      cadena += ` ${matriz[fil][col]} `;
    }
    console.log(cadena);
  }
}
```



Matrices

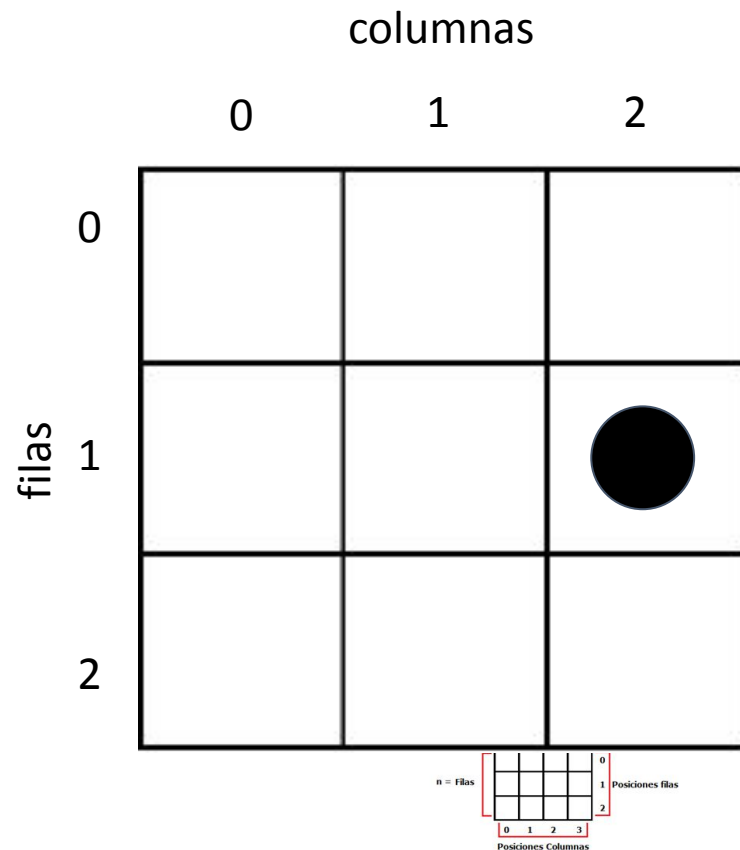
¿Cómo accedo a un valor dentro de la matriz?

Si queremos acceder al valor (circulo negro) guardado en un elemento de la matriz podemos acceder a el de la siguiente manera:

```
let valor:tipo=nombreMatriz[fila][columna];
```

Ej:

```
let valor:number=matriz[1][2];
```



Técnicas de Programación

Carrera Programador full-stack

Matrices (Ejercicios)

Matrices

Crear una matriz cuadrada (5x5) donde cada fila impar de la matriz son ceros y el resto son unos.

Luego un usuario ingresará una posición de la matriz, (un valor para una fila y otro para la columna). En esa posición deberá reemplazar el valor existente por un 5.

Ej. de cómo se debería ver la salida

```
[ 1, 1, 1, 1, 1 ]  
[ 0, 0, 0, 0, 0 ]  
[ 1, 5, 1, 1, 1 ]  
[ 0, 0, 0, 0, 0 ]  
[ 1, 1, 1, 1, 1 ]
```