ESCUELA DE INGENIERÍA EN CIENCIAS Y SISTEMAS
FACULTAD DE INGENIERÍA
UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

ECYS | DIDT
DESARROLLO DE TRANSFERENCIA TECNOLÓGICA

Día, Fecha:	Miércoles, 01/02/2022
Hora de inicio:	07:10

Introducción a la programación y computación 1 [D]

José Eduardo Morales García

Introducción a la Programación y Computación 1

Contenido

1. **Introducción memoria estática**
2. Vectores
3. Matrices
4. Fundamentos de programación
5. Estructuras de control
6. Métodos(Procedimientos)
7. Funciones
8. Recursividad

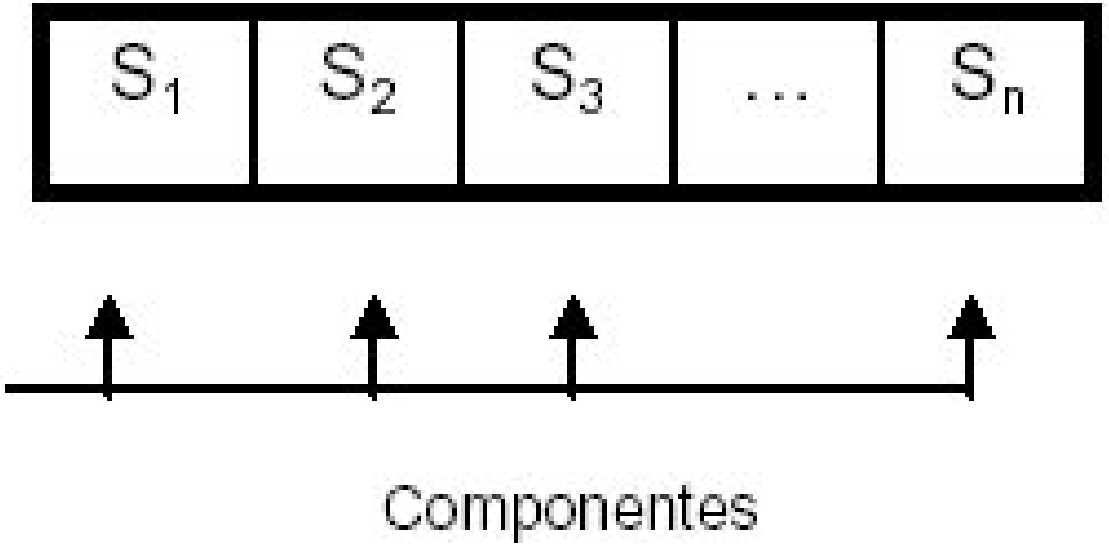
1. Introducción

Los vectores y matrices en programación son espacios de almacenamiento que contienen una serie de elementos del mismo tipo.

Contenido

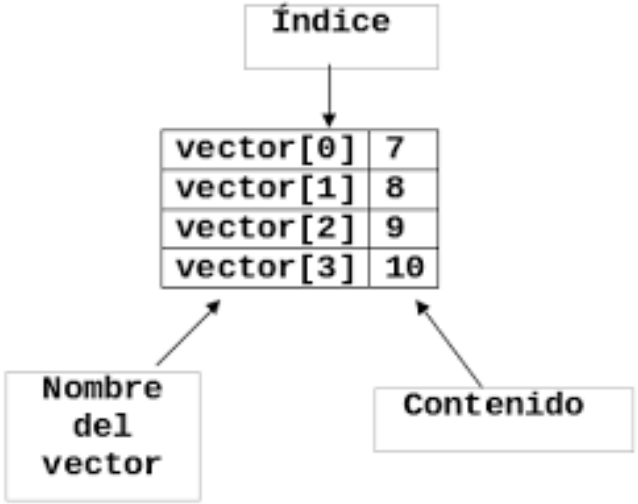
1. Introducción memoria estática
2. **Vectores**
3. Matrices
4. Fundamentos de programación
5. Estructuras de control
6. Métodos(Procedimientos)
7. Funciones
8. Recursividad

2. Vectores



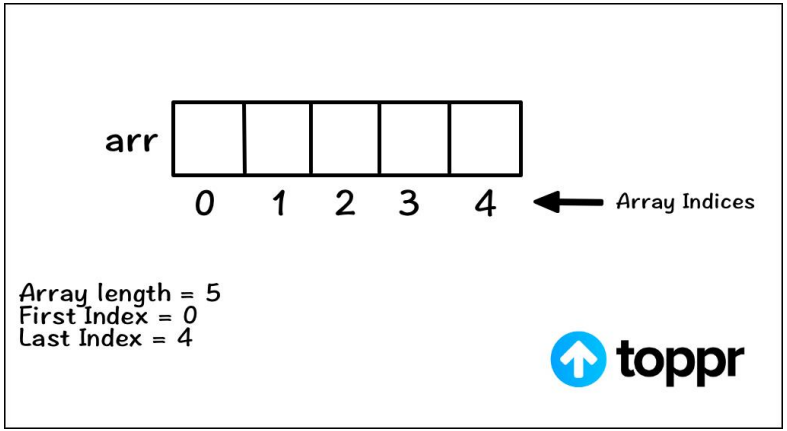
2. Vectores

Es un tipo de estructura unidimensional d
contiene una serie de datos contiguos d



2. Vectores

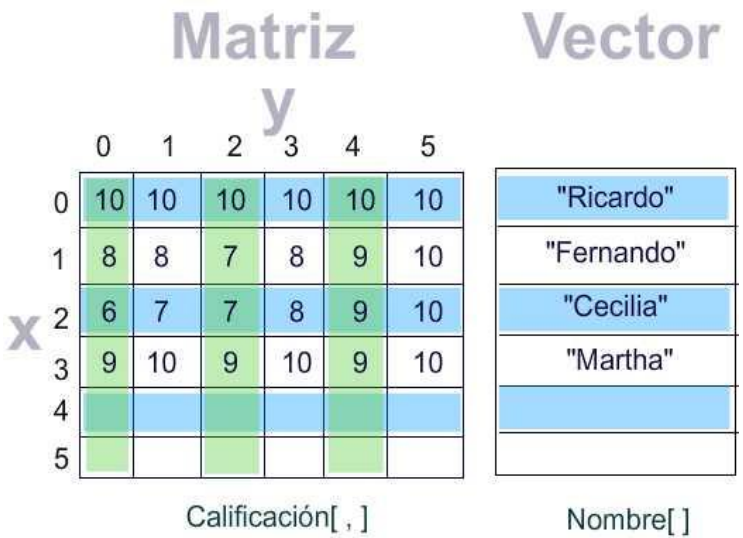
Un vector esta definido por el tipo de dato que se desea representar, pueden ser definidos por tipos primitivos y no primitivos.



Contenido

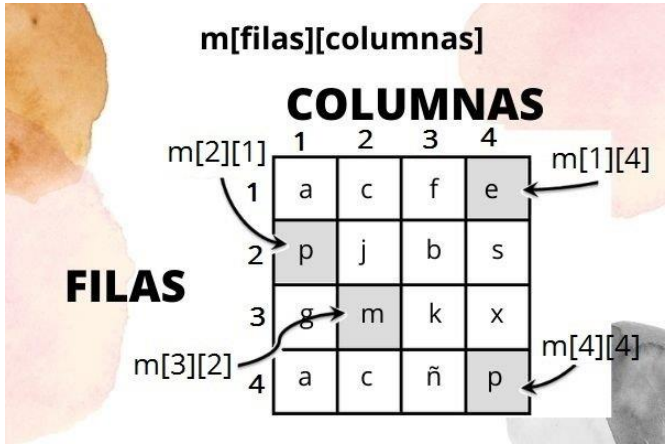
1. Introducción memoria estática
2. Vectores
3. **Matrices**
4. Fundamentos de programación
5. Estructuras de control
6. Métodos(Procedimientos)
7. Funciones
8. Recursividad

3. Matrices


Matriz
Vector
y
x
0 1 2 3 4 5
0 10 10 10 10 10 10
1 8 8 7 8 9 10
2 6 7 7 8 9 10
3 9 10 9 10 9 10
4
5
"Ricardo"
"Fernando"
"Cecilia"
"Martha"
Calificación[,]
Nombre[ ]

3. Matrices

Una matriz por su parte es un conjunto o colección de arreglos, es decir es un vector de vectores.



3. Matrices

Son regularmente empleados, para almacenar conjuntos de valores, de un mismo tipo, en el que podemos ordenar y agrupar conforme a lo requiramos.

Calendario[semana, nombredia] nxm

nombredia

Semana	D	L	m	M	J	V	S
1	0	0	0	0	1	2	3
2	4	5	6	7	8	9	10
3	11	12	13	14	15	16	17
4	18	19	20	21	22	23	24
5	25	26	27	28	0	0	0

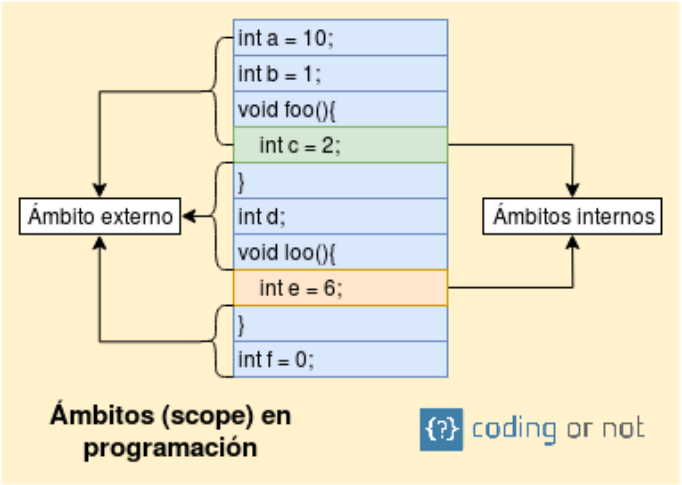
Contenido

1. Introducción memoria estática
2. Vectores
3. Matrices
4. **Fundamentos de programación**
5. Estructuras de control
6. Métodos(Procedimientos)
7. Funciones
8. Recursividad

4. Fundamentos de programación

Ámbito, es el contexto otorgado conforme a los permisos dentro de un programa. (“donde es accesible un valor dentro del programa”)

- **Global**
- **Local**



4. Fundamentos de programación

Ámbito local, únicamente es accesible dentro del bloque al que pertenece

Ámbito Global, accesible desde cualquier lugar

```
main() {
    int a = 1;                                   B1
    int b = 1;
    {
        int b = 2;                               B2
        {
            int a = 3;                           B3
            cout << a << b;
        }
        {
            int b = 4;                           B4
            cout << a << b;
        }
        cout << a << b;
    }
    cout << a << b;
}
```

4. Fundamentos de programación

Operadores

- Unarios
- Aritméticos
- Relacionales
- Condicionales
- Bits
- Ternario
- Desplazamiento

Descripción	Operadores
operadores posfijos	op++ op--
operadores unarios	++op --op +op -op ~ !
multiplicación y división	* / %
suma y resta	+ -
desplazamiento	<< >> >>>
operadores relacionales	< > <= =>
equivalencia	== !=
operador AND	&
operador XOR	^
operador OR	|
AND booleano	&&
OR booleano	||
condicional	?:
operadores de asignación	= += -= *= /= %= &= ^= |= <<= >>= >>>=

4. Fundamentos de programación

Palabras reservados, una palabra con un significado y valor especial para el lenguaje de programación en el cual se trabaja.

4. Fundamentos de programación

abstract	continue	finally	int	public	throw
assert	default	float	interface	return	throws
boolean	do	for	long	short	transient
break	double	goto	native	static	true
byte	else	if	new	strictfp	try
case	enum	implements	null	super	void
catch	extends	import	package	switch	volatile
class	false	inner	private	synchronized	
const	final	instanceof	protected	this	while

Contenido

1. Introducción memoria estática
2. Vectores
3. Matrices
4. Fundamentos de programación
5. **Estructuras de control**
6. Métodos(Procedimientos)
7. Funciones
8. Recursividad

5. Estructuras de control

```
Proceso Adivina_Numero

    intentos<-10
    num_secreto <- azar(100)+1

    Escribir "Adivine el numero (de 1 a 100):"
    Leer num_ingresado
    Mientras num_secreto<>num_ingresado Y intentos>1 Hacer
        Si num_secreto>num_ingresado Entonces
            Escribir "Muy bajo"
        Sino
            Escribir "Muy alto"
        FinSi
        intentos <- intentos-1
        Escribir "Le quedan ",intentos," intentos:"
        Leer num_ingresado
    FinMientras

    Si num_secreto=num_ingresado Entonces
        Escribir "Exacto! Usted adivino en ",11-intentos," intentos."
    Sino
        Escribir "El numero era: ",num_secreto
    FinSi

FinProceso
```

5. Estructuras de control

Las **estructuras de control,** son aquellas que determinan el comportamiento de un programa.

- **Selectivas,** permiten que la ejecución de un programa o conjunto de instrucciones se ejecuten conforme una condición o criterio.
- **Iterativas,** permiten la ejecución de un bloque de código o conjunto de instrucciones de forma repetitiva.

5. Estructuras de control

Las **estructuras de control selectivas:**

1. **If, if-else**
2. **switch**

5. Estructuras de control

Estructura de Selección IF,

Permiten la ejecución de un bloque si cumple con dos únicas alternativas, **verdadero** o **falso.**

****es la estructura de selección principal.**

5. Estructuras de control

```java
if(carrera == "Ingenieria en Ciencias y Sistemas"){
  //...
  System.out.println("La mejor carrera del mundo");
}
// -- esta es opcional, se utiliza conforme a lo que requerimos conforme a nuestro flujo
else {
  //...
  System.out.println("La mejor carrera del mundo es Ingenieria en Ciencias y Sistemas");
}
```

5. Estructuras de control

Estructura de Selección SWITCH,

Esta sentencia se utiliza para elegir una entre múltiples opciones, únicamente evalúa valores puntuales.

5. Estructuras de control

```
switch(opcion)
  {
      case "1" -> {
          //caso si el valor ingresado es "1"
      }
      case "2" -> {
          //caso si el valor ingresado es "2"
      }
      case "3" -> {
         //caso si el valor ingresado es "3"
      }
      default  -> {
        //caso si el valor ingresado no forma parte en ninguna de las anteriores
      }
  }
```

5. Estructuras de control

Estructura Iterativa WHILE,

Esta sentencia se utiliza como un ciclo de ejecución, en el cual al cumplirse cierta condición repetirá los bloques hasta que esta deje de cumplir con la misma.

5. Estructuras de control

```
while(continues)
{
  /*
        Esta operara siempre el bloque repetitivamente
        hasta que continues represente un falso logico

  */
  System.out.println("Repetir");
}
```

5. Estructuras de control

Estructura Iterativa FOR,

Esta sentencia se utiliza como un ciclo de ejecución, en el cual se ejecuta un numero determinado de veces dentro de su ciclo de ejecución.

5. Estructuras de control

```java
for(int a=0; a<15; a++)
{
     /*
        Esta operara siempre el bloque repetitivamente
        hasta que a sea un numero igual a 15

    */
    System.out.println("imprimiendo el valor " + a + " veces");
}
```

5. Estructuras de control

Estructura Iterativa DO…WHILE,

Esta sentencia se utiliza como un ciclo de ejecución, en el cual se ejecuta para desarrollar un conjunto de instrucciones al menos una vez o varias veces.

5. Estructuras de control

```
Do
{
  /*
        Esta operara siempre el bloque 1 vez, luego
        lo realizara repetitivamente hasta que continues
        represente un falso logico

  */
  System.out.println("Repetir")
}while(continues)
```

Contenido

1. Introducción memoria estática
2. Vectores
3. Matrices
4. Fundamentos de programación
5. Estructuras de control
6. **Métodos(Procedimientos)**
7. Funciones
8. Recursividad

6. Procedimientos

Un **procedimiento**, o **método** es un conjunto de instrucciones como un mini programa que se ejecuta dentro de un programa, a solicitud.

- **Parámetros**, un parámetro en si es un objeto o valor que sirve como entrada o sirve para la ejecución y construcción de un valor como salida.
- **Bloque de instrucción,** es el conjunto de instrucciones contenidas dentro del mismo, la conforman diversos tipos de instrucciones entre ellas las estructuras de control

6. Procedimientos

```java
// un metodo puede contener o no parametros
public void suma(int a, int b)
{
  /*

    Dentro de un metodo pueden venir diversos conjuntos
    de instrucciones o bloques.

  */
  int suma = a + b;
  System.out.println("El valor de la suma es: " + suma);

}
```

Contenido

1. Introducción memoria estática
2. Vectores
3. Matrices
4. Fundamentos de programación
5. Estructuras de control
6. Métodos(Procedimientos)
7. **Funciones**
8. Recursividad

7. Funciones

Una **función** es un conjunto de instrucciones como un mini programa que se ejecuta dentro de un programa, a solicitud, las cuales a diferencia de los métodos nos retornaran un valor conforme al tipo de dato que representa.

- **Parámetros**, un parámetro en si es un objeto o valor que sirve como entrada o sirve para la ejecución y construcción de un valor como salida.
- **Bloque de instrucción,** es el conjunto de instrucciones contenidas dentro del mismo, la conforman diversos tipos de instrucciones entre ellas las estructuras de control

7. Funciones

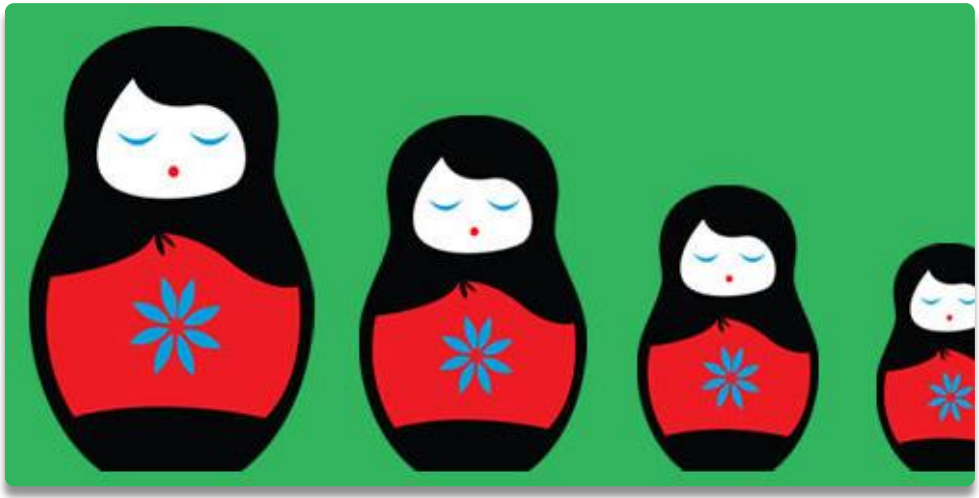
```java
// una funcion puede contener o no parametros
public int suma(int a, int b)
{
  /*

    Dentro de una funcion pueden venir diversos conjuntos
    de instrucciones o bloques.

  */
  int suma = a + b;
  System.out.println("El valor de la suma es: " + suma);
  return suma;
}
```

8. Recursividad

- La recursividad es un concepto fundamental en matemáticas y en computación.
- Es una alternativa diferente para implementar estructuras de repetición (ciclos). Los módulos se hacen llamadas recursivas.

8. Recursividad

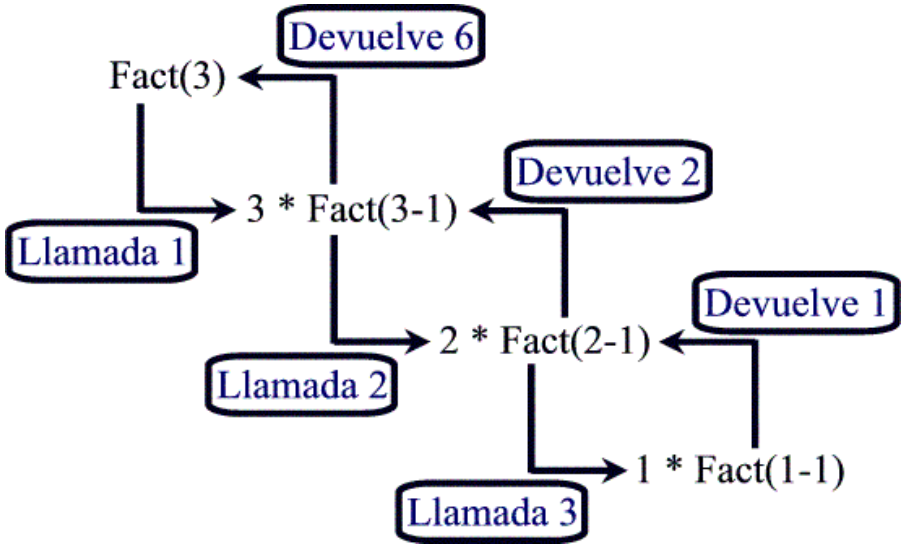
- La recursividad se compone de uno o varios casos base.
- Cada caso base es una solucion simple



8. Recursividad

- Caso recursivo: una solución que involucra volver a utilizar la función original, con parámetros que se acercan más al caso base.
- Los pasos que sigue el caso recursivo son los siguientes:
  - 1. El procedimiento se llama a sí mismo
  - 2. El problema se resuelve, tratando el mismo problema, pero de tamaño menor
  - 3. La manera en la cual el tamaño del problema disminuye asegura que el caso base eventualmente se alcanzará

8. Recursividad



Parte practica