Introducción a la Programación y Computación 1

Contenido

1. **Introducción**
2. El Lenguaje
3. Compilador e Interprete
4. Fundamentos de programación
5. Estructuras de control
6. Procedimientos
7. Funciones

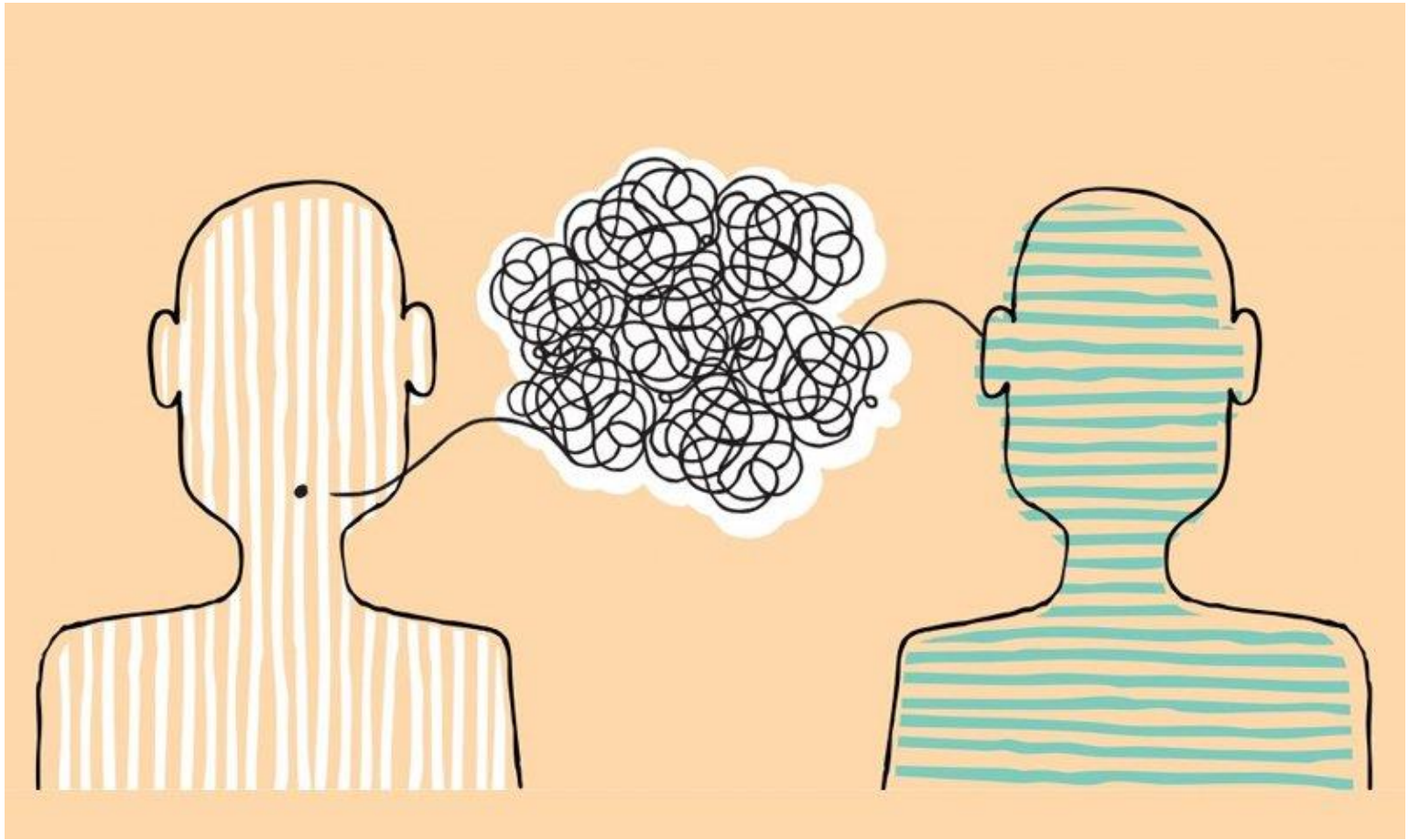
1. Introducción

Durante la unidad 2, usted aprenderá acerca de los conceptos básicos que fundamentan la programación en cualquier lenguaje de programación, aunque este se encuentre orientado al idioma JAVA, cabe resaltar que la lógica de programación es el mismo en todos.

Contenido

1. Introducción
2. **El Lenguaje**
3. Compilador e Interprete
4. Fundamentos de programación
5. Estructuras de control
6. Procedimientos
7. Funciones

2. El lenguaje



2. El lenguaje

La Real Academia de la Lengua define el lenguaje como la "facultad del ser humano de expresarse y comunicarse con los demás a través del sonido articulado o de otros sistemas de signos".

Lenguaje de programación, es un conjunto de símbolos y reglas sintácticas, léxicas y semánticas, que definen una estructura y significado en cada uno de sus elementos, el cual es utilizado para controlar el comportamiento lógico y físico de una maquina.

2. El lenguaje

1. **Lenguaje maquina,** es el lenguaje primitivo que es entendible por el ordenador.
2. **Lenguaje de bajo nivel**, son lenguajes que abarcan el funcionamiento de la maquina, en este lenguaje se trabaja de manera directa con la memoria de la computadora.
3. **Lenguaje de alto nivel,** es el lenguaje principal que interactúa con el usuario, es decir es el lenguaje como tal.

Contenido

1. Introducción
2. El Lenguaje
3. **Compilador e Interprete**
4. Fundamentos de programación
5. Estructuras de control
6. Procedimientos
7. Funciones

3. Compilador e interprete

```
package main

import "fmt"

func main() {
  fmt.Printf("hello, world")
}
```

```
01010101111011100011010
01000101000101010010100
01010100101010100000101
00110100010101000111100
01100101001010101010010
11100011010100100100001
```

3. Compilador e interprete

Compilador, es el encargado de traducir el lenguaje de alto nivel (entendible para el desarrollador) a un lenguaje de maquina.

Interprete, es un programa capaz de analizar y ejecutar otro programa, el cual va ejecutando línea por línea.

3. Compilador e interprete

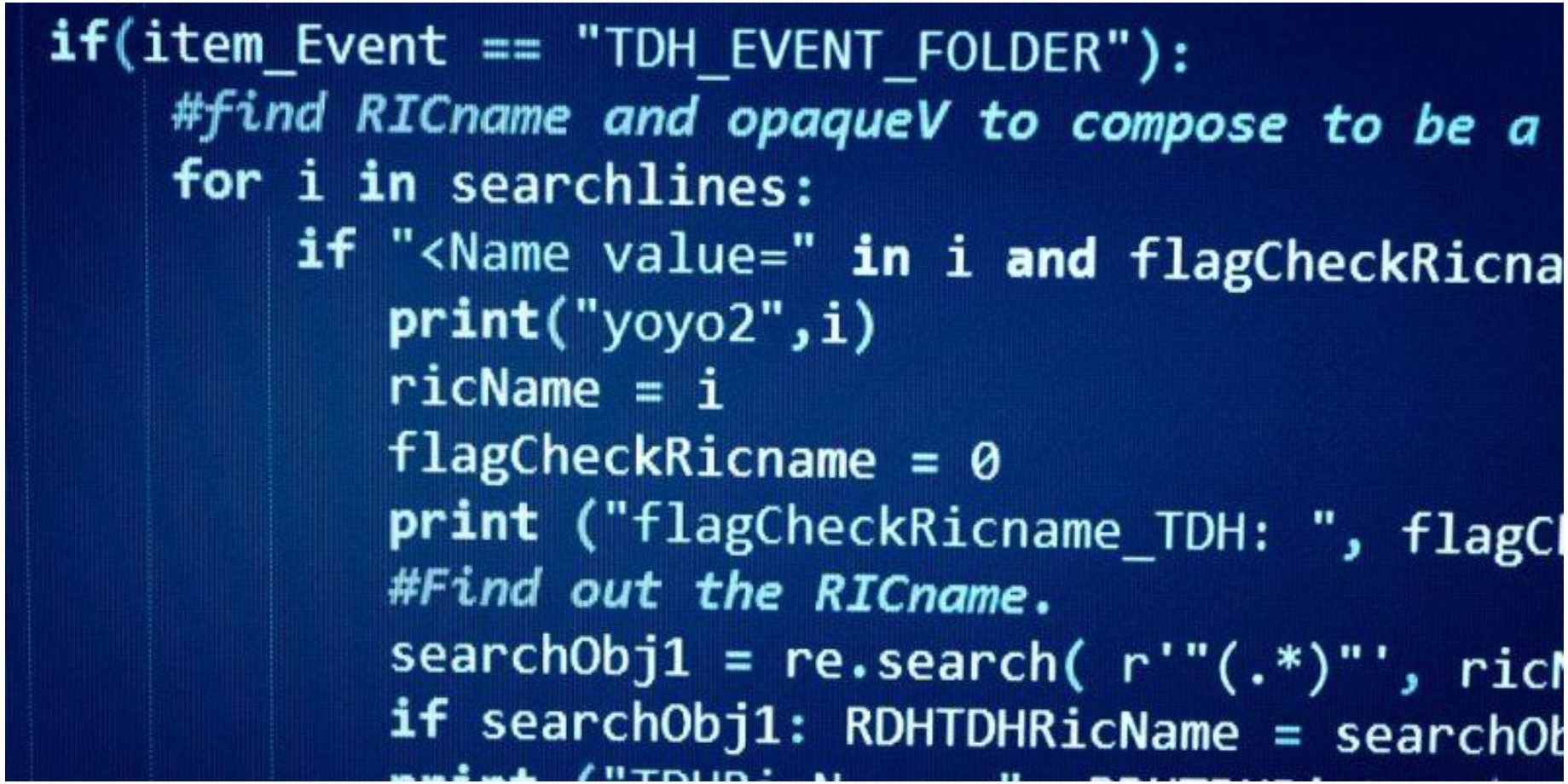
Diferencias de un Compilador y un Intérprete

Compilador	Intérprete
Tiene como salida un único lenguaje objeto.	Tiene como salida instrucciones traducidas
El rendimiento en la ejecución del programa compilado (la salida) es más rápido que interpretado.	El rendimiento (del intérprete) se somete al rendimiento de analizar la traducción una a una.
La salida puede depender de la arquitectura.	Tiende a ser más portable e independiente de la arquitectura.
No requiere del programa fuente porque el programa objeto es ejecutable. **Puede ser secreto**(El programa fuente)	Se requiere del lenguaje fuente para su ejecución.
Los errores sintácticos y semánticos se detectan antes de la ejecución del programa objeto.	Se detectan los errores en la ejecución del programa.
Tiene menos flexibilidad en el uso de la memoria para el programa objeto.	Es más flexible para que el programa pueda usar la memoria.

Contenido

1. Introducción
2. El Lenguaje
3. Compilador e Interprete
4. **Fundamentos de programación**
5. Estructuras de control
6. Procedimientos
7. Funciones

4. Fundamentos de programación



4. Fundamentos de programación

Programación modular, es aquella en la cual se divide el programa en módulos o programas para que sea mas legible y manejable

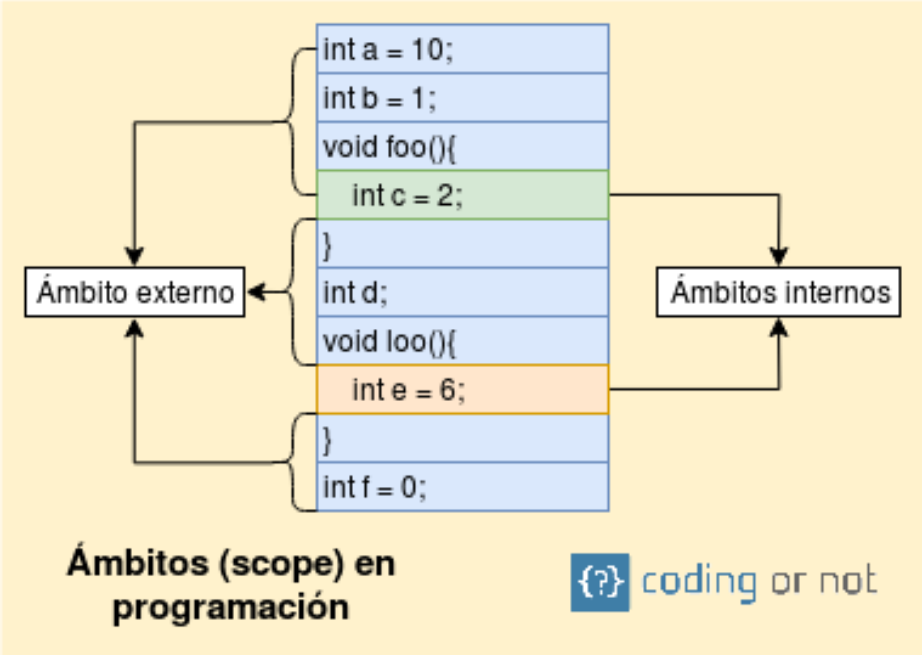
Programación estructurada, orientada a mejorar la claridad, calidad y tiempo de desarrollo de un software usando estructuras de control básicas:

1. **Estructura secuencial**
2. **Estructura selectiva**
3. **Estructura iterativa**

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

Identificadores, nombre definido para denotar un elemento, es decir es la forma de ubicar un valor dentro del programa.

- Únicamente se define por lenguaje alfanumérico.
- No emplear letras especiales como ñ, letras tildadas, entre otros.
- Se puede utilizar “_”
- No se puede iniciar con un numero, se debe iniciar con letra o “_”

****consejo,** utilizar idioma ingles ya que al ser un lenguaje global es entendible por todos**.**

4. Fundamentos de programación

Tipos de datos, todo lenguaje de programación lo conforma la clasificación de datos dentro de su sintaxis, regularmente se dividen en **primitivos** y **no primitivos**.

- **Primitivo,** son los tipos de datos originales o elementales dentro de un lenguaje de programación.
- **No primitivo**, lo conforman los tipos de objetos especiales dentro del mismo.

4. Fundamentos de programación

		NOMBRE	TIPO	OCUPA	RANGO APROXIMADO
TIPOS DE DATOS EN JAVA	**TIPOS PRIMITIVOS** (sin métodos; no son objetos; no necesitan una invocación para ser creados)	byte	Entero	1 byte	-128 a 127
		short	Entero	2 bytes	-32768 a 32767
		int	Entero	4 bytes	$2*10^9$
		long	Entero	8 bytes	Muy grande
		float	Decimal simple	4 bytes	Muy grande
		double	Decimal doble	8 bytes	Muy grande
		char	Carácter simple	2 bytes	---
		boolean	Valor true o false	1 byte	---
	TIPOS OBJETO (con métodos, necesitan una invocación para ser creados)	Tipos de la biblioteca estándar de Java	String (cadenas de texto) Muchos otros (p.ej. Scanner, TreeSet, ArrayList...)		
		Tipos definidos por el programador / usuario	Cualquiera que se nos ocurra, por ejemplo Taxi, Autobus, Tranvia		
		arrays	Serie de elementos o formación tipo vector o matriz. Lo consideraremos un objeto especial que carece de métodos.		
		Tipos envoltorio o wrapper (Equivalentes a los tipos primitivos pero como objetos.)	Byte		
			Short		
			Integer		
			Long		
			Float		
			Double		
			Character		
			Boolean		

4. Fundamentos de programación

Tipo	Representación / Valor	Tamaño (en bits)	Valor mínimo	Valor máximo	Valor por defecto
`boolean`	`true` o `false`	1	N.A.	N.A.	`false`
`char`	Carácter Unicode	16	\u0000	\uFFFF	\u0000
`byte`	Entero con signo	8	-128	128	0
`short`	Entero con signo	16	-32768	32767	0
`int`	Entero con signo	32	-2147483648	2147483647	0
`long`	Entero con signo	64	-9223372036854775808	9223372036854775807	0
`float`	Coma flotante de precisión simple Norma IEEE 754	32	±3.40282347E+38	±1.40239846E-45	0.0
`double`	Coma flotante de precisión doble Norma IEEE 754	64	±1.79769313486231570E+308	±4.94065645841246544E-324	0.0

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
2. El Lenguaje
3. Compilador e Interprete
4. Fundamentos de programación
5. **Estructuras de control**
6. Procedimientos
7. Funciones

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
2. El Lenguaje
3. Compilador e Interprete
4. Fundamentos de programación
5. Estructuras de control
6. **Procedimientos**
7. Funciones

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
2. El Lenguaje
3. Compilador e Interprete
4. Fundamentos de programación
5. Estructuras de control
6. Procedimientos
7. **Funciones**

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;
}
```