

Tema 1



INTRODUCCIÓN A LA PROGRAMACIÓN

1. ¿QUÉ ES PROGRAMAR?



- **Programar** significa planificar y concretar la secuencia de órdenes concisas para la realización de una tarea.
- Cuando informáticamente se habla de programar se está pensando en las órdenes que se deben dar al ordenador para que lleve a cabo una tarea concreta.
- Esto se hace a través de lo que se denomina **programa**.

2. ¿QUÉ ES PROGRAMA?



- Un **programa informático** es un conjunto de instrucciones que una vez ejecutadas realizarán una o varias tareas en un ordenador.
- Al conjunto general de programas informáticos se le denomina **software**.

2. ¿QUÉ ES ALGORITMO?



- Cuando se plantea un problema que debe solucionarse mediante un programa, una buena técnica de trabajo es analizar y representar de forma detallada las operaciones que se llevarán a cabo para solucionar el problema.
- **Algoritmo** es la representación detallada de las operaciones que se llevarán a cabo para solucionar un programa.
- Un mismo problema puede resolverse con más de un algoritmo y por tanto más de un programa.

2. ¿QUÉ ES ALGORITMO?



- Todo algoritmo debe ser:
 - **Preciso:** orden de operaciones
 - **Finito:** en el número de operaciones
 - **Correcto:** debe solucionar el problema
- Estas tres condiciones son imprescindibles en cualquier programa, pero además hay que tener en cuenta la **eficiencia** (líneas de código y tiempo de ejecución) que hace que de dos programas diferentes que resuelven correctamente un problema, uno sea mejor que otro.

3. Lenguajes de Programación



- Para construir programas informáticos se utilizan los **lenguajes de programación**.
- Un lenguaje de programación está formado por un vocabulario y por un conjunto de reglas gramaticales que permiten escribir órdenes que una computadora llevará a cabo.
- Cada lenguaje tiene sus reglas y su vocabulario propio, si bien existe un conjunto de elementos fundamentales que la mayoría de ellos comparte (estructuras de control, comentarios, etc).

3.1. Generaciones



● **Primera Generación:** son conocidos como **lenguajes máquina**. Se trata de un conjunto de códigos numéricos binarios que se introducen directamente en la máquina para que ésta, sin mediar traducción, ejecutase las órdenes del programador. Cada máquina, tiene su lenguaje máquina, lo que obligaba a los antiguos programadores a conocer tantos de estos lenguajes como máquinas quisieran programar.

Ejemplo: 00000 00000 000010 00000 00100
000000

3.1. Generaciones



⦿ **Segunda Generación:** son conocidos como **lenguajes ensamblador**. Pueden ser leídos por un programador, aunque su estructura sintáctica aún dista bastante del lenguaje humano. Para que un programa escrito en un lenguaje de este tipo pueda ejecutarse, primero debe de traducirse a código máquina.

3.1. Generaciones



EJEMPLO:

```
; HOLA.ASM
; Programa clasico de ejemplo. Despliega una leyenda en pantalla.
STACK  SEGMENT STACK                ; Segmento de pila
        DW  64 DUP (?)              ; Define espacio en la pila
STACK  ENDS

DATA  SEGMENT                      ; Segmento de datos
SALUDO  DB  "Hola mundo!!",13,10,"$" ; Cadena
DATA  ENDS

CODE  SEGMENT                      ; Segmento deCodigo
        ASSUME CS:CODE, DS:DATA, SS:STACK

INICIO:                                ; Punto de entrada al programa
        MOV  AX,DATA                ; Pone direccion en AX
        MOV  DS,AX                  ; Pone la direccion en los registros
        MOV  DX,OFFSET SALUDO        ; Obtiene direccion del mensaje
        MOV  AH,09H                  ; Funcion: Visualizar cadena
        INT  21H                     ; Servicio: Funciones alto nivel DOS
        MOV  AH,4CH                  ; Funcion: Terminar
        INT  21H
CODE  ENDS
        END  INICIO                  ; Marca fin y define INICIO
        MOV  AH,4CH
```

3.1. Generaciones



- **Tercera Generación:** aparecieron a finales de la década de 1950 por la necesidad de acercar la forma de comunicación de los humanos a la comunicación con las computadoras. Lenguajes como Fortran, Cobol o Algol fueron los primeros en desarrollar una sintaxis similar a la de los lenguajes humanos, incorporando por primera vez la posibilidad de declarar tipos, variables, emplear estructuras de control, etc. La mayoría de los lenguajes de programación más populares, como C, C++, Java, Ada o Basic son lenguajes de tercera generación.

Ejemplo:

```
for (i=0;i<10;i++)  
    System.out.println(i);
```

3.1. Generaciones



- ◉ **Cuarta Generación:** son los más cercanos a la forma de expresarse y comunicar de los humanos. El programador no expresa directamente cómo debe hacer las cosas la computadora, sino sólo qué es lo que se debe hacer. Suelen catalogarse como lenguajes de 4GL todo lenguaje que ha sido diseñado y creado con un propósito específico, a menudo relacionado con las consultas y accesos a bases de datos. SQL, PowerBuilder, NATURAL son algunos ejemplos.

Ejemplo:

FIND ALL FLIGHTS WHERE DESTINATION IS “MADRID”

3.1. Generaciones



- ◉ **Quinta Generación:** son aquellos orientados a especificar en una computadora el problema a resolver, pero no cómo resolverlo. A menudo relacionados con la inteligencia artificial, su diseño permite especificar el problema a resolver mediante un conjunto de restricciones, sin necesidad de escribir un algoritmo que guíe su resolución.

Se utilizan mucho en la investigación (en áreas como el procesamiento natural o los sistemas expertos). Prolog, Lisp, Mercury son algunos ejemplos.

Ejemplo:

`suspende(X) <=noEstudia(X) and haceExamen(X)`

4. Tipos de código



- ◉ Si abrimos un programa con un editor de texto aparecen símbolos ininteligibles (para nosotros).
- ◉ Los programas están en lenguaje binario (**lenguaje máquina**), que es el que entienden las máquinas.
- ◉ Nosotros aprenderemos a escribir programas en un lenguaje entendible para los humanos (**lenguajes de alto nivel**) .
- ◉ Nosotros escribimos el código de un programa en un lenguaje determinado (**código fuente**), esas instrucciones son traducidas al lenguaje máquina (**código máquina**) por otros programas llamados **compiladores** o **intérpretes**

4. Tipos de código



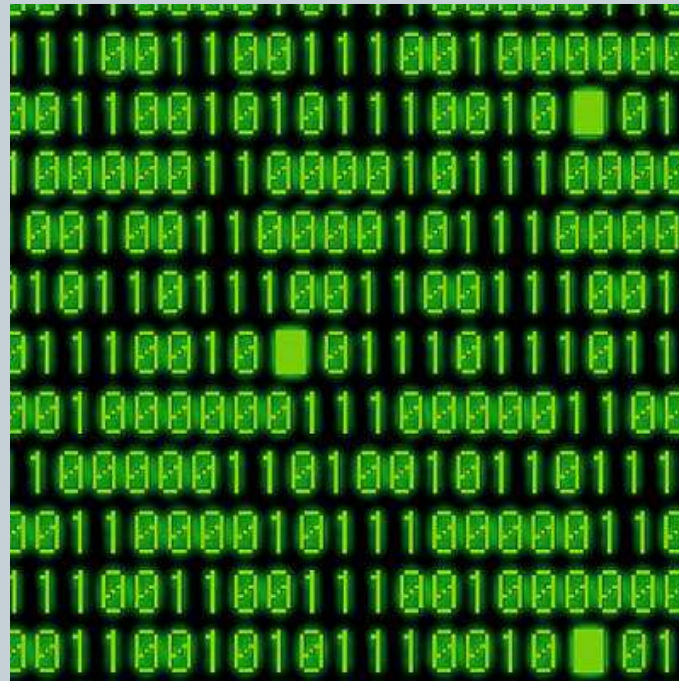
⦿ **Código fuente:** es el escrito directamente por los programadores en editores de texto, lo cual genera el programa. Contiene el conjunto de instrucciones codificadas en algún lenguaje de alto nivel (JAVA, C, etc).

```
public class HelloWorld {  
  
    /**  
     * @param args  
     */  
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
        System.out.println("Hello world!");  
    }  
}
```

4. Tipos de código



- ⦿ **Código objeto:** es el código binario resultante de procesar con un compilador el código fuente.



4. Tipos de código



⦿ **Código ejecutable:** Es el código binario resultado de enlazar uno o más fragmentos de código objeto con las rutinas y bibliotecas necesarias.

Constituye uno o más archivos binarios con un formato tal que el sistema operativo es capaz de cargarlo en la memoria RAM, y proceder a su ejecución directa.

Por lo tanto se dice que el código ejecutable es directamente «inteligible por la computadora».

4. Tipos de código



5. Proceso de traducción

- El proceso de obtención del código ejecutable a partir de código fuente es el siguiente:



6. Herramientas de desarrollo



- ◉ Las herramientas de desarrollo son aquellos programas o aplicaciones que tienen cierta importancia en el desarrollo de un programa.
- ◉ **Editor:** programa que permite crear y modificar programas escritos en lenguaje de alto nivel. Ejemplo: Scite, GNU Emacs, vi.
- ◉ **Compilador:** es un programa que permite traducir el código fuente de un programa en lenguaje de alto nivel, a otro lenguaje de nivel inferior (típicamente lenguaje de máquina). Ejemplo: gcc, g++, javac.

6. Herramientas de desarrollo



- ◉ **Enlazador (linker):** es un programa que toma los objetos generados en los primeros pasos del proceso de compilación, la información de todos los recursos necesarios (biblioteca), quita aquellos recursos que no necesita, y enlaza el código objeto con su(s) biblioteca(s) con lo que finalmente produce un fichero ejecutable. Ejemplo: ld.
- ◉ **Ensamblador:** son aquellos programas que se encargan de desestructurar el código en lenguaje ensamblador y traducirlo a lenguaje binario. Los archivos en lenguaje binario serán posteriormente enlazados en un único fichero, el ejecutable. Ejemplo: tas, gas

6. Herramientas de desarrollo



- ◉ **Depuradores:** sirve para corregir bugs. Se encargan de ejecutar un programa paso a paso para ir advirtiéndolo de errores, valores de variables, etc. Muy útiles cuando el programa parece estar bien, pero no da el resultado esperado (se cuelga, da resultados erróneos...). Ejemplo: GDB, jdb
- ◉ **IDEs (Entornos de desarrollo integrado):** juntan en un sólo programa editor de texto, compilador, enlazador, ensamblador, depurador... Ejemplos de ellas son Anjuta, Dev-Cpp, Codeblocks, Eclipse, Netbeans, etc.

7. Compilación



- El proceso de compilación consta de las siguientes fases:
 - **Análisis léxico:** en esta fase se lee el programa fuente de izquierda a derecha y se comprueba que los símbolos del lenguaje se han escrito correctamente. Además, se eliminan los espacios en blanco, los comentarios, etc.
 - **Análisis sintáctico:** en esta fase se comprueba si lo obtenido de la fase anterior es sintácticamente correcto, es decir, cumple con la gramática del lenguaje.

7. Compilación



- **Análisis semántico:** revisa el programa fuente para tratar de encontrar errores semánticos.
Ejemplo: si cada operador tiene operandos permitidos por la especificación del lenguaje fuente.
- **Síntesis:** Consiste en generar el código objeto equivalente al programa fuente. Sólo se genera código objeto cuando el programa fuente está libre de errores de análisis, lo cual no quiere decir que el programa se ejecute correctamente, ya que un programa puede tener errores de concepto o expresiones mal calculadas.

8. Compilación vs Interpretación



- ◉ Un **intérprete** es un programa informático capaz de analizar y ejecutar programas escritos en un lenguaje de alto nivel.
- ◉ Se diferencian de los compiladores en que mientras estos analizan todo el programa dejando un programa libre de errores para después ejecutarlo, los intérpretes van analizando instrucción a instrucción para comprobar si son correctas o no. Si una instrucción no es correcta se finaliza la ejecución del programa en ese punto, mientras que si es correcta inmediatamente se ejecuta.
- ◉ Los programas interpretados suelen ser más lentos que los compilados debido a la necesidad de traducir el programa mientras se ejecuta, pero a cambio son más flexibles como entornos de programación y depuración.

8. Compilación vs Interpretación



- **Analogía:** un compilador equivale a un traductor profesional que, a partir de un texto, prepara otro independiente traducido a otra lengua, mientras que un intérprete corresponde al intérprete humano, que traduce de viva voz las palabras que oye, sin dejar constancia por escrito.

9. Errores



- Cuando codificamos un programa pueden darse distintos tipos de errores
 - **Sintácticos:** los detecta el compilador, son fáciles de corregir, se refieren a algo mal escrito.
 - **De ejecución:** el programa es correcto sintácticamente pero no acaba su ejecución de forma correcta.
 - **De lógica:** el programa si acaba de ejecutarse pero no procesa como debe.

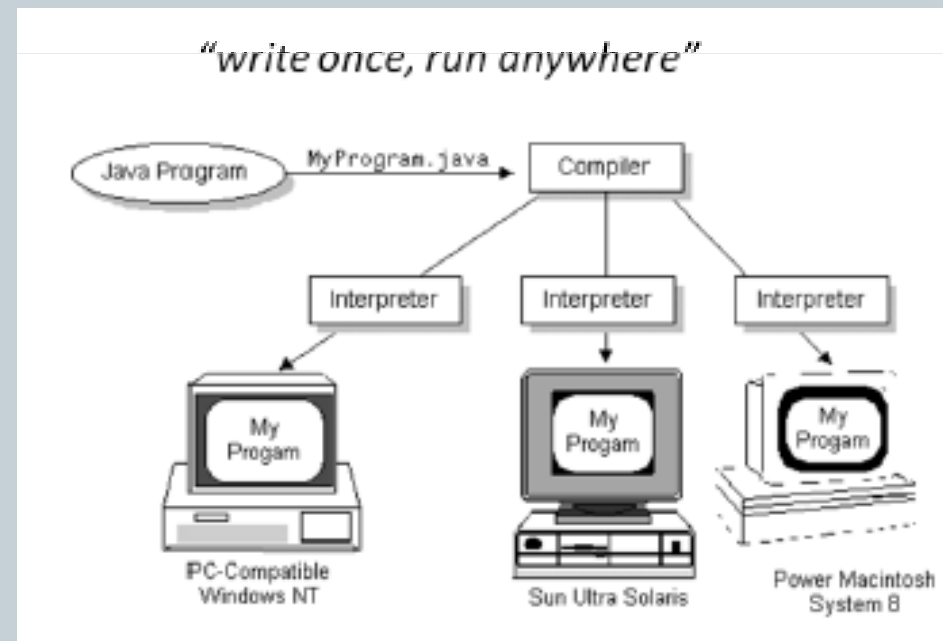
10. El lenguaje Java



- **Java** es uno de los lenguajes más utilizados en la actualidad.
- Es un lenguaje de propósito general y multiplataforma.
- Diseñado en 1990 por James Gosling.
- Su primer nombre fue OAK y tuvo como referente C y C++.
- Java utiliza una máquina virtual en el sistema destino y por lo tanto no hace falta recompilar de nuevo la aplicación para cada sistema operativo.

10. El lenguaje Java

- Por lo tanto, Java es un lenguaje compilado e interpretado (híbrido) que utiliza un código intermedio (**bytecode**) independiente de la arquitectura y por lo tanto puede ser ejecutado en cualquier sistema (dibujo).



10. El lenguaje Java



- Como se puede apreciar en el dibujo, en Java, una vez compilado el programa, se puede ejecutar en cualquier plataforma solamente con tener instalada la máquina virtual (JVM) de Java.
- Sin embargo, en un lenguaje compilado como C, deberemos recompilar el programa para el sistema destino con la consiguiente pérdida de flexibilidad.
- En Java, el compilador compila a bytecode y el intérprete se encargará de ejecutar ese código intermedio en la máquina real.

10. 1El JDK



- El JDK (Java Development Kit) no contiene ninguna herramienta gráfica para el desarrollo de programas pero sí contiene aplicaciones de consola y herramientas de compilación, documentación y depuración.
- El JDK incluye el JRE (Java Runtime Environment) que consta de los mínimos componentes necesarios para ejecutar una aplicación Java, como son la máquina virtual y las librerías de clases.
- El JDK contiene, entre otras, las siguientes herramientas de consola:
 - Java: es la máquina virtual de Java

10. 1El JDK



- Javac: es el compilador de Java
 - Jdb: es el depurador de consola de Java
 - Javadoc: es el generador de documentación
- Descarga el jdk de la página web:
<http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>
 - A continuación, modifica el valor de la variable de entorno **PATH** y **CLASSPATH** para añadirle donde está situado el directorio bin del JDK y la ruta donde se encuentran las clases JAVA al ejecutar.

10. 2 Estructura de un programa



- Los programas Java se componen de unos ficheros .java (contienen el código fuente).
- Dichos ficheros se compilan y se construyen los ficheros equivalentes .class (bytecode).
- La aplicación se ejecuta desde el método principal main() situado en una clase.
- Dicha clase es la clase principal (por ella se empieza a ejecutar el programa).

```
public class Holamundo{  
  
    public static void main(String [] args){  
  
        .....  
    }  
}
```


10. 2 Estructura de un programa



- Las primeras líneas que forman parte de un programa suelen estar compuestas de comentarios acerca del nombre del programa, el programador, la fecha de creación, observaciones...
- Aparecen con `//` si lo que viene a continuación es texto en una sola línea o con `/* */` cuando se trata de más de una línea.
- Hay en Java otro tipo de comentario `/**` comentario de documentación, de una o más líneas `*/` -> Javadoc

10.3 Tipos de datos simples



- Los tipos de variables de Java ocupan siempre la misma memoria y tienen el mismo rango de valores en cualquier tipo de ordenador.

- **Enteros:**

byte	8 bits	[-128,127]
short	16 bits	[-32768,32767]
int	32 bits	[-2147483648,2147483647]
long	64 bits	aprox 10 exp 18

- **Reales:** en coma flotante

float	32 bits	aprox 10 exp 38
double	64 bits	aprox 10 exp 308
Por ejemplo:	3.14	2e12 3.1E12

10. 3 Tipos de datos simples



- **Booleanos:** boolean

Toman dos valores: true (verdadero) o false (falso). Ocupan un bit.

```
boolean esMenorDeEdad, esPensionista;  
int edad;
```

- **Caracteres:** char

Incluyen prácticamente los caracteres de todos los idiomas. Ocupan 16 bits.

10.4 Variables



- Zona de memoria donde se almacena información del tipo de dato que desee el programador.
- Su declaración consiste en definir el nombre de la misma con su tipo correspondiente.
- Además se pueden inicializar en la declaración.

```
Tipo nombre;  
Tipo nombre1, nombre2,...;  
Tipo nombre=valor;
```

- Los nombres deben empezar por una letra o por el carácter _ o \$, después pueden combinar letras y números.

10.4 Variables



- Se distinguen las mayúsculas de las minúsculas y no hay longitud máxima.
- Por convenio empiezan por minúscula y si contienen varias palabras, se unen y todas las palabras excepto la primera empiezan por mayúscula.

```
int x=0;  
float y =3.5;  
char letra='a';
```

10. 4.1 Visibilidad de una variable



- La visibilidad o ámbito de una variable es la parte del código de una aplicación donde la variable es accesible y puede ser utilizada.
- Las variables se declaran dentro de un bloque (por bloque se entiende el contenido entre las llaves {}) y son accesibles sólo dentro de ese bloque.

```
public class Ejemplo {  
  
    static int n1=50; //variable miembro de la clase  
  
    public static void main (String [] args){  
  
        int n2=30, suma=0; //variables locales  
        suma=n1+n2;  
        System.out.println("La suma es: " + suma);  
    }  
}
```

10. 4.1 Visibilidad de una variable



- Las variables declaradas en el bloque de la clase como n1 se consideran miembros de la clase (**variable global**), mientras que las variables n2 y suma pertenecen al método main y sólo pueden ser utilizados en el mismo.
- Las variables declaradas en el bloque de código de un método son variables que se crean cuando el bloque se declara, y se destruyen cuando finaliza la ejecución de dicho bloque. -> **Variable Local**
- Las variables globales se inicializan por defecto (las numéricas con 0 y los caracteres con '\0', mientras que las variables locales no se inicializan por defecto.

10.5 Constantes



- Si en la declaración de una variable se pone por delante la palabra reservada “**final**”, significa que se trata de una constante, es decir que no se puede modificar su valor a lo largo del programa.

```
final double PI=3.1415;
```

- Por convenio, se declaran en mayúsculas.

10. 6 Palabras clave



- Las siguientes son las palabras clave que están definidas en Java y que no se pueden utilizar como identificadores:

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

10. 7 Operadores



- Los operadores son símbolos que indican como se van a manipular los datos (operandos).
- Se clasifican en:
 - **Operadores aritméticos:**

Suma: +
Resta: -
Producto: *
División: /
Módulo o resto de una división de números enteros (no para float); %

- **Operadores relacionales:**

Mayor: >	Mayor o igual: > =
Menor: <	Menor o igual: < =
Igual: ==	Distinto: !=

10. 7 Operadores



- **Operadores lógicos:** son necesarios para describir condiciones en las que aparecen conjunciones –y- (&&), disyunciones –o- (||) y negaciones (!)

```
boolean trabaja, esUnTrabajadorJoven, esUnVotante;  
esUnTrabajadorJoven = esMenorDeEdad & trabaja;  
esUnVotante = !esMenorDeEdad;
```

- **Operadores de incremento y decremento:** son dos operaciones típicas para aumentar o disminuir en una unidad el valor de una variable contador. Son ++ y --.

```
int a, b;  
a = 5;  
b = 7;  
a = b++ : asigno y sumo a=7, b=8  
a = ++b : sumo y asigno b=8, a=8  
  
a++ equivale a de a = a + 1 . Igual que ++a
```

10. 7 Operadores



○ Operadores de asignación:

=	A = B	Asignación.
*=	A *= B	Multiplicación y asignación. La operación A*=B equivale a A=A*B.
/=	A /= B	División y asignación. La operación A/=B equivale a A=A/B.
%=	A %= B	Módulo y asignación. La operación A%=B equivale a A=A%B.
+=	A += B	Suma y asignación. La operación A+=B equivale a A=A+B.
-=	A -= B	Resta y asignación. La operación A-=B equivale a A=A-B.

Nociones básicas para realizar ejercicios



- **Salida por pantalla**

```
System.out.println(elementos)
```

```
System.out.print(elementos)
```

- Se diferencian en que println salta de línea después de escribir.
- Si tiene como elemento una cadena de caracteres, se escribe delimitada por comillas. Si en el interior lleva comillas se escribe \".

```
System.out.print("El dijo \"No");
```

Nociones básicas para realizar ejercicios



- Si la cadena de caracteres es demasiado larga para que coja en una línea partimos la cadena y usamos el signo +.

```
System.out.println("Esta cadena" +  
                    " se escribe" +  
                    " en tres líneas");
```

- Para imprimir números o variables:

```
Int s=80;  
  
System.out.println("Voy a escribir una var:" + s + "y un número" + 30*15);
```

11. Pensar primero y programar después



- Vamos a resolver el problema de ir al cine a ver una película.
- Para ello, debemos elaborar un algoritmo que resuelva dicho problema.
- Un algoritmo consta de:
 - Datos de entrada
 - Operaciones sobre dichos datos
 - Datos de salida
- Análisis del problema:
 - Datos de entrada: nombre de la película, dirección del cine, hora de proyección.
 - Datos de salida: ver la película.

11. Pensar primero y programar después



- **Análisis del problema:**
 - Datos auxiliares: entrada, número de butaca
 - **Diseño del algoritmo:**
 - Inicio**
 - //Seleccionar película**
 - Conectarse a Internet**
 - Mientras no lleguemos a la cartelera**
 - Buscar en Google**
 - Mientras no se acabe la cartelera**
 - Leer argumento película**
 - Si nos gusta, recordarla**
 - Elegir una de las películas seleccionadas**

11. Pensar primero y programar después



- **Análisis del problema:**

//Comprar la entrada online

Hacer clic en la película elegida

Elegir día y horario de la película

Si no queda sitio en la sala o no nos gusta el sitio, volver a elegir horario de la película

Comprar la entrada con la tarjeta de crédito

Leer la dirección del cine y la hora de proyección

//Recoger entrada

Trasladarse al cine con nuestra entrada

Si hay cola

Ponerse el último

Mientras no lleguemos a la taquilla

Avanzar

11. Pensar primero y programar después



- **Análisis del problema:**

Recoger la entrada

Si tenemos hambre, comprar palomitas

//Ver la película

Leer el número de sala y número de butaca

Buscar la butaca

Sentarse

Fin

11. Pensar primero y programar después



- Diseñar un algoritmo que resuelva el problema de fregar los platos de la comida.
- Análisis del problema:
 - Datos de entrada: platos sucios.
 - Datos de salida: platos limpios.
 - Datos auxiliares: número de platos que quedan
 - **Diseño del algoritmo**

Inicio

Abrir el grifo

Tomar un estropajo

Echarle jabón

11. Pensar primero y programar después



Mientras queden platos

Lavar plato

Aclararlo

Colocarlo en el escurridor

Mientras queden platos en el escurridor

Secar plato

Fin

11. Pensar primero y programar después



- Diseñar un algoritmo que resuelva el problema de hacer una taza de té.
- Análisis del problema:
 - Datos de entrada: bolsa de té, agua.
 - Datos de salida: taza de té.
 - Datos auxiliares: sonido de la tetera, aspecto de la infusión, sabor de la infusión, leche, azúcar
 - **Diseño del algoritmo**

Inicio

Llenar de agua la tetera

Conectar la tetera

Mientras no hierva el agua

Esperar

11. Pensar primero y programar después



Tomar la bolsa de té

Introducir la bolsa de té en la taza

Verter el agua hirviendo hasta la mitad de la taza

Mientras no esté hecho el té

Esperar

Quitar la bolsa de té

Verter leche en la taza

Mientras no tenga el aspecto deseado

Seguir vertiendo leche en la taza

Probar el té

Mientras no tenga el sabor deseado

Echar azúcar en la taza

Probar el té

Fin

11. Pensar primero y programar después



- Por lo tanto, si queremos diseñar un algoritmo, lo primero es **identificar las tareas más importantes** necesarias para resolver el problema y ordenarlas según su ejecución.
- Los pasos de este primer boceto se verificarán, añadiéndoles más detalles; incluso en algunas ocasiones necesitarán ser **refinados una vez más antes** de conseguir un algoritmo **claro, perfectamente preciso y completo**.
- A este método de diseño de algoritmos por medio de etapas, pasando de los conceptos generales a los concretos con refinamientos sucesivos se le llama **método descendente (top down)**.

11.1 Diagramas de flujo



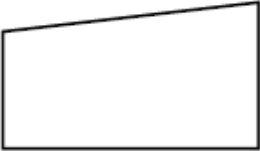







- Esta técnica se usa para representar visual y claramente el flujo de datos y la secuencia de ejecución de los programas.
- Estos diagramas pueden representar varios niveles de concreción de los programas.
- Los que indican el flujo de datos de un sistema sin entrar en detalles se llaman **organigramas**.
- Los que detallan la secuencia de acciones de un algoritmo se denominan **ordinogramas**.

11.1.1 Organigramas









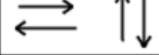





- Los organigramas disponen de símbolos que representan los soportes, es decir, las entradas, las salidas y los dispositivos de almacenamiento.
- Algunos de los símbolos, los podemos ver en la siguiente tabla:

Proceso:	Entrada de teclado:	Salida por impresora:	Disco magnético:
			
Pantalla:	Cinta magnética:	Tarjeta perforada:	Tambor magnético:
			

11.1.2 Ordinogramas

- Los **ordinogramas** son los diagramas más útiles para representar los algoritmos con detalle.
- En ellos se usan los símbolos, que podemos ver en la siguiente tabla:

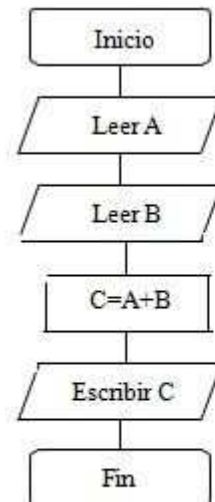
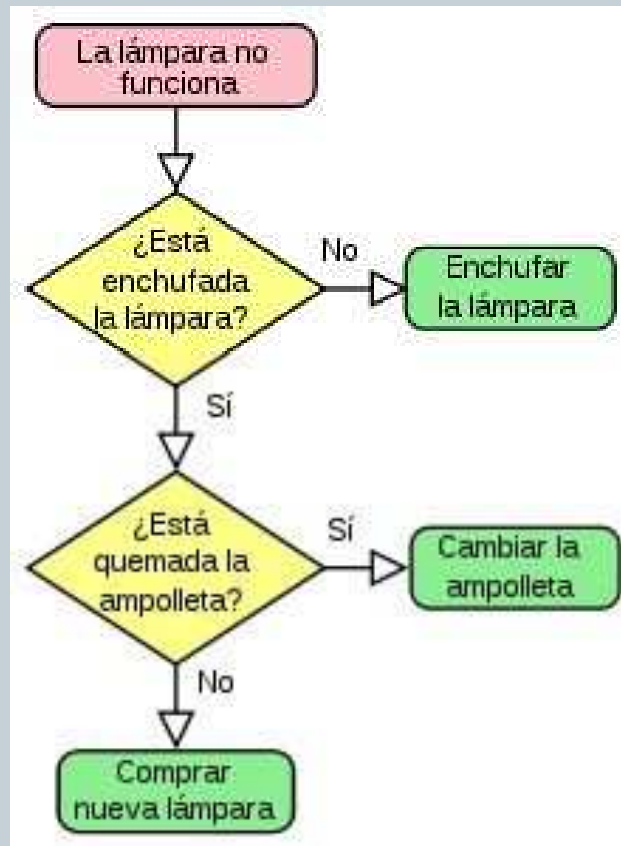
	Inicio o fin del programa
	Pasos, procesos o líneas de instrucción de programa de computo
	Operaciones de entrada y salida
	Toma de decisiones y Ramificación
	Conector para unir el flujo a otra parte del diagrama
	Cinta magnética
	Disco magnético
	Conector de página
	Líneas de flujo
	Anotación
	Display, para mostrar datos
	Envía datos a la impresora

11.1.3 Normas de construcción



- Los diagramas de flujo se deben escribir de arriba hacia abajo o de izquierda a derecha.
- Se debe evitar el cruce de líneas; si no hay mas opciones usaremos conectores.
- Todos los objetos deberán estar conectados por líneas de flujo.
- En el caso de las alternativas, deben haber mas de una línea de salida, para así indicar el camino que hay que seguir según la decisión que se tome.
- El texto que utilicemos en las formas debe ser breve y legible.

11.1.4 Ejemplos



→ Leemos el primer número y lo dejamos en A

→ Leemos el segundo número y lo dejamos en B

→ Sumamos A y B, y dejamos el resultado en C

→ Escribimos C

www.areatecnologia.com

11.1.4 Ejemplos

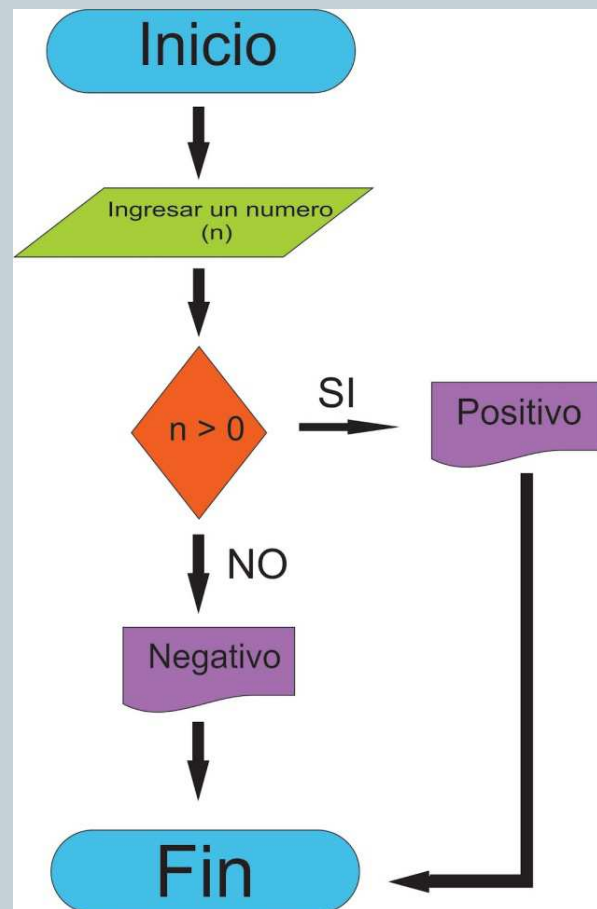


Diagrama de flujo que indica si un número es positivo o negativo.

11.2 Pseudocódigo



- Es una técnica que posibilita la inscripción de los algoritmos como si fuera un lenguaje de programación, pero usando palabras del lenguaje natural.
- De esta manera el programador se centra en la resolución del problema y no en su sintaxis.
- Este lenguaje es muy flexible, así que no necesita una estructura estándar; aunque es obligado seguir unas normas sobre estructura y coherencia.

11.2 Pseudocódigo



- Estas son las instrucciones que se suelen usar:
 - **Primitivas:** Pueden ser de salida con la instrucción escribir, de entrada o leer y de asignación. Para este propósito se puede usar el signo igual = o la flecha → en primer lugar la variable destino, y después la variable o el valor explícito de origen.
 - **Sentencias de control:** Las **alternativas** son escritas según la sintaxis SI condición ENTONCES, a continuación las instrucciones que se ejecutan si se cumple la condición cuando termina FIN-SI. Las sentencias **alternativas dobles** tienen intercalada la sentencia SINO.

11.2 Pseudocódigo



- **Ejemplo:**

SI esta soso el arroz ENTONCES

Echar sal

SINO SI esta salado ENTONCES

Echar agua

FIN SI

- Se podrían expresar las sentencias de control **repetitivas** como MIENTRAS condición HACER instrucciones FIN-MIENTRAS. o REPETIR instrucciones HASTA condición o PARA variable DE un valor inicial A un valor limite incremento (tipo de incremento del valor de la variable de control) luego las instrucciones y por último el FIN-PARA.

11.2 Pseudocódigo



○ Ejemplo:

MIENTRAS arroz soso HACER

 Echar sal

FIN MIENTRAS

REPETIR

 Echar sal

HASTA arrozOK

12. Paradigmas de programación



- Con el tiempo se han establecido diferentes criterios y técnicas de programación, no excluyentes entre sí:
 - **Programación modular:** consta de varias secciones o módulos que interactúan a través de llamadas a procedimientos. El módulo principal, encargado de llamar al resto de los módulos, integra el programa en su totalidad.
 - **Programación estructurada:** está compuesta por un conjunto de técnicas evolucionadas que aumentan la productividad del programa. Se basa en un proceso lineal y sencillo y se apoya en estructuras secuenciales, selectivas y repetitivas.
 - **Programación orientada a objetos:** usa objetos y sus interacciones para diseñar aplicaciones. Los objetos son entidades que poseen una serie de propiedades destinadas a que los programas sean más fáciles de escribir, mantener y reutilizar.

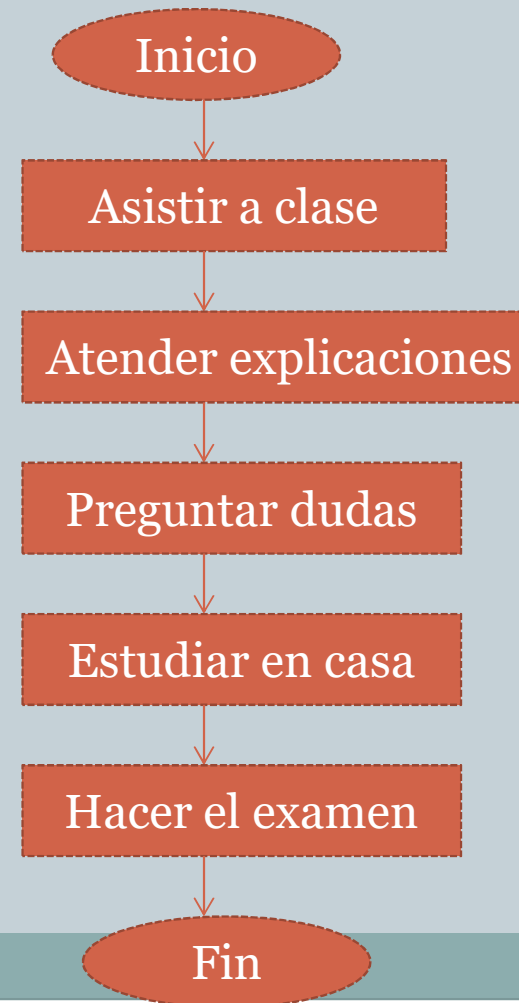
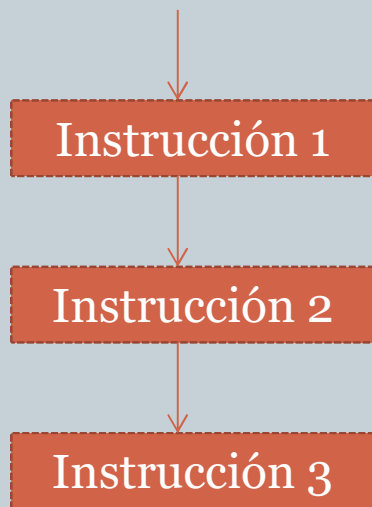
12.1 La programación estructurada



- **Lenguajes estructurados** son aquellos en los que las sentencias incluidas en el código se ejecutan de forma lineal y sencilla (una detrás de otra) a partir de un solo punto de entrada (inicio) y otro de salida (final).
- Estos lenguajes se fundamentan en una teoría que demuestra que todo programa se puede escribir utilizando únicamente tres tipos de estructuras de control: secuenciales, condicionales y repetitivas.

10.1 La programación estructurada

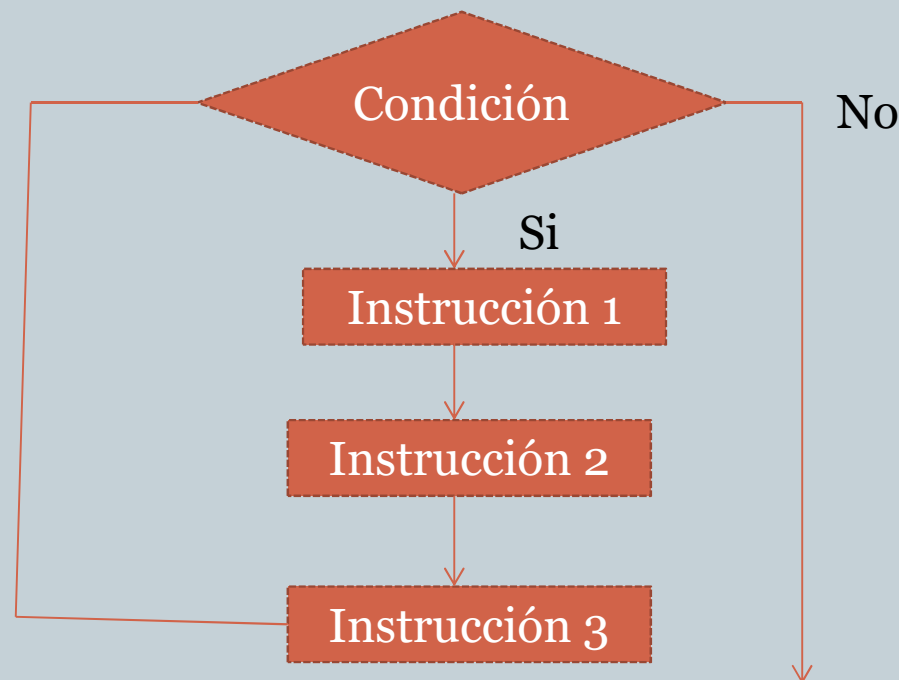
- **Secuenciales:** son un bloque de instrucciones sucesivas que se ejecutan de forma ordenada y seguida.



10.1 La programación estructurada

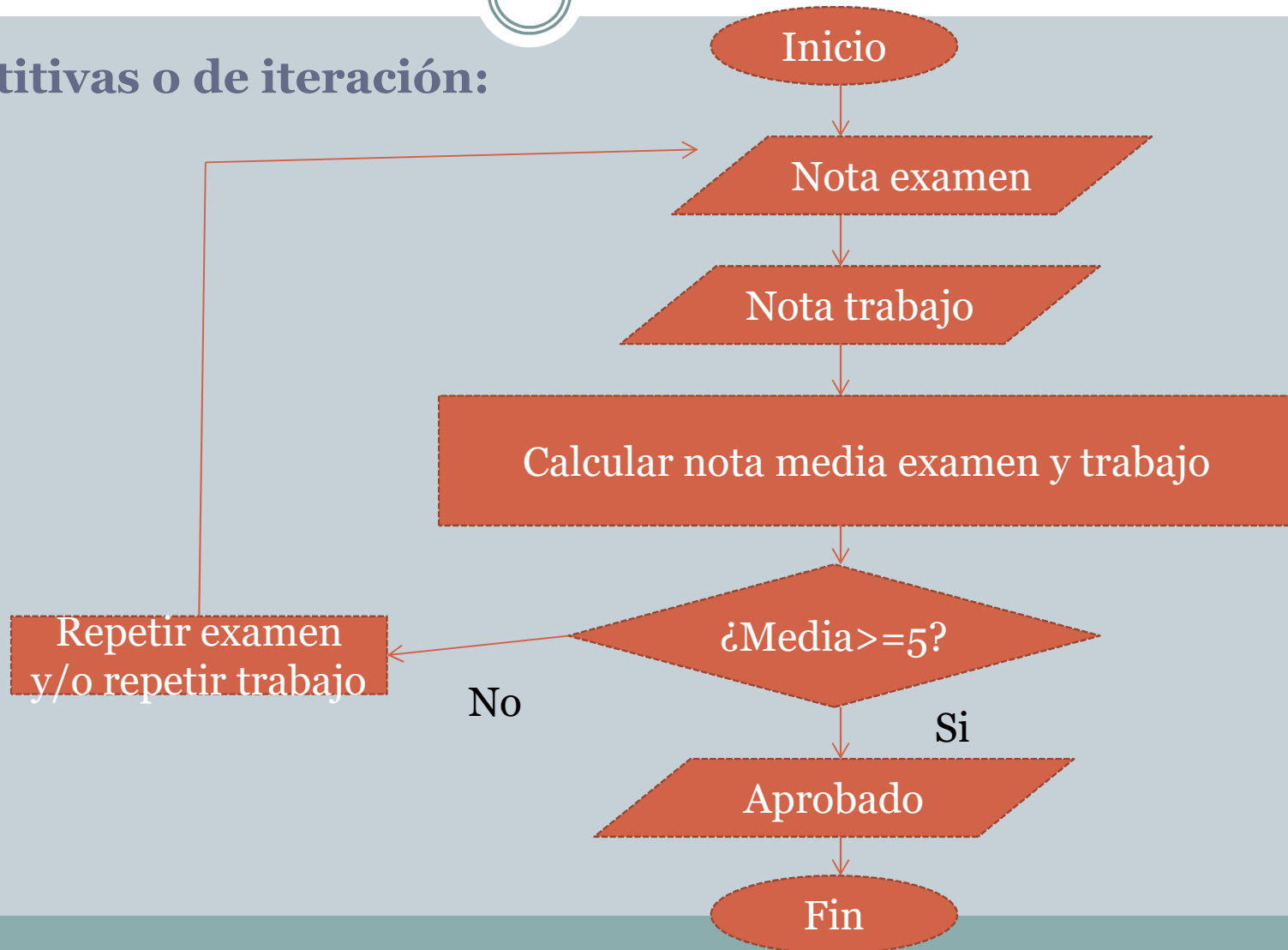


- **Repetitivas o de iteración:** Son instrucciones que se repiten un número limitado de veces o hasta que se cumple una determinada condición (mientras <condicion> haz <instrucción>; en inglés: while <condicion> do <instrucción>)



10.1 La programación estructurada

○ Repetitivas o de iteración:



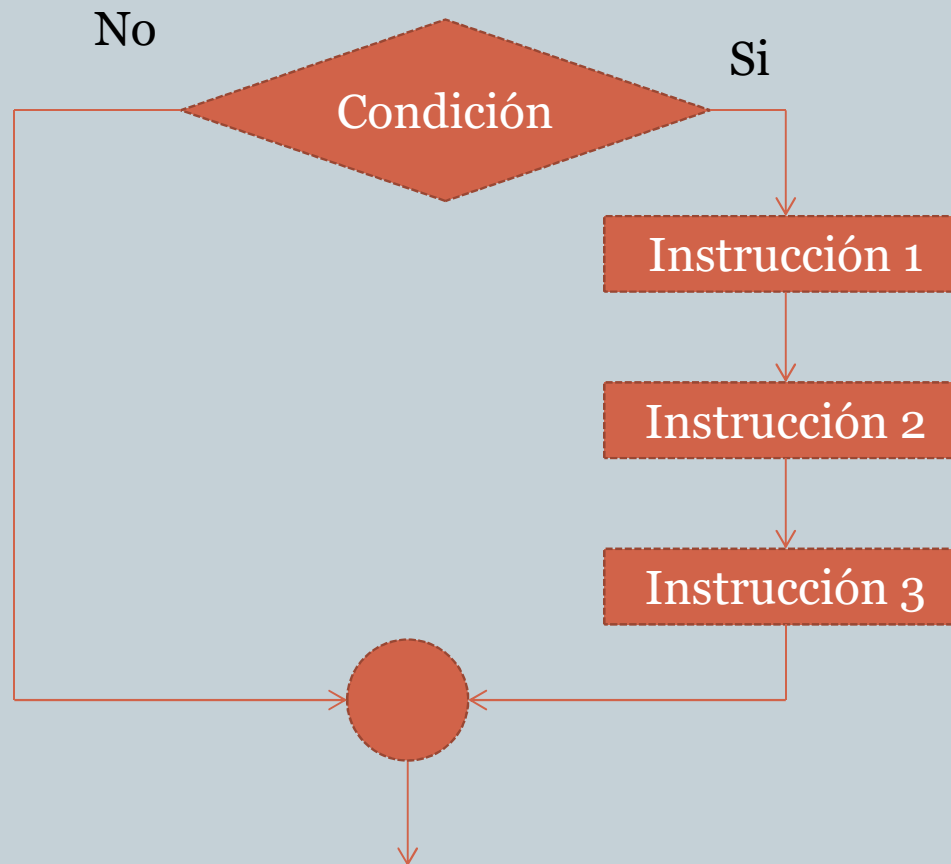
10.1 La programación estructurada



- **Condicionales o selectivas:** son instrucciones que permiten establecer condiciones. En función de si éstas se cumplen o no, se ejecutan unas instrucciones u otras (si <condicion> entonces <instrucción 1> en caso contrario <instrucción 2> o, según la terminología inglesa: if <condicion> then <instrucción 1> else <instrucción 2>)

10.1 La programación estructurada

○ Condicionales o selectivas:



Actividad



Realiza los ejercicios de la hoja de ejercicios Diagramas de Flujo y Pseudocódigo